75.61 – Taller de Programacion III

Trabajo Práctico 1: Yet Another Apache Monitor

1er cuatrimestre 2018

2da entrega – 12/04/18

Padron: 95470

Nombre: Gabriel Gayoso

Email: ga-yo-so@hotmail.com

Facultad de Ingeniería

Universidad de Buenos Aires

Introducción

Este trabajo consiste en el diseño, desarrollo y testeo de un monitor de mensajes de log para servidores Apache. Como base se tienen una serie de requerimientos que se deben cumplir:

- Requerimientos Funcionales:
  - Se debe monitorear un archivo de log de servidores Apache redirigido a la entrada estándar de la aplicación.
  - Es necesario relevar las líneas de log de forma constante para identificar estadísticas con: cantidad de requests por segundo, requests por cliente, cantidad de errores, 10 recursos más pedidos.
  - Se deben identificar los mensajes de error con más repeticiones.
  - La aplicación se debe detener si se detecta un cierre del archivo de log o la señal de interrupción control+C.
- Requerimientos No Funcionales:
  - La aplicación debe priorizar la recepción de la entrada de logs en todo momento y ejecutar el análisis en paralelo.
  - Utilizar cantidades controladas de hilos (revisar necesidad de *Thread Pools*)
  - Se deben mostrar las estadísticas por pantalla con una frecuencia de 1 minuto.
  - Se debe realizar una descarga de todas las entradas de log recibidas a un archivo.
  - Se debe realizar una descarga de los mensajes de error en otro archivo de texto, incluyendo únicamente fecha y mensaje.
  - El almacenamiento de datos y entidades del sistema se puede realizar utilizando algún esquema simple de serialización (no es foco del TP el uso eficiente de la persistencia).

Suposiciones

Se relevaron los requerimientos mencionados para anotar aquellas áreas bien definidas que tendrán un impacto directo sobre el diseño e implementación, y otras más ambiguas sobre las que fue necesario definir una estrategia a seguir.

Suposición 1 – Existe un formato conocido que tendrán los mensajes de log recibidos

La aplicación se centra en el monitoreo de mensajes de log de servidores Apache. Estos mensajes son altamente configurables, por lo que se definió una estructura para los mensajes de la forma: “client datetime [log_type] “METHOD request http_ver” status_code status_msg”. Por ejemplo:

- 192.168.0.1 28/03/2018:16:05:33 [error] “GET /home/test/file HTTP/1.1” 200 Ok
- 168.14.22.1 23/06/1952:13:23:42 [log] “POST /home/asd/1 HTTP/1.1” 404 File not found

Suposición 2 – La ventana para la visualización de estadísticas temporales no es estricta

Uno de los requerimientos es la visualización de estadísticas calculadas en una ventana de tiempo. Dado que el servidor probablemente este encendido todo el tiempo, estará enviando mensajes de log constantemente a la aplicación. Estas estadísticas entonces serán informativas del estado general del servidor en un momento dado, pero dicho momento no necesita estar definido rigurosamente. Por esta razón no considero necesario hacer un corte fuerte para el cálculo de cada estadística. Es suficiente con realizar un cálculo aproximado aún si se procesa algún mensaje extra durante el cálculo de ellas.

Suposición 3 – La cantidad de estadísticas a acumular no es muy elevada

Existe una distinción entre estadísticas y los valores utilizados para calcularlas. Por ejemplo, para el cálculo de “cantidad de requests por cliente” es necesario conocer la cantidad de requests y la cantidad de clientes. Estos valores podrían acumularse en un contador, pero si en un futuro se quisiera agregar la estadística “los 10 clientes con más requests realizadas”, sería necesario agregar dicha estadística. Para evitar esto, propongo separar por un lado el almacenamiento de datos estadísticos, y por otro el procesamiento de ellos para obtener ciertas estadísticas deseadas. Esto implica que como máximo será necesario acumular apariciones para cada uno de los campos definidos en la Suposición 1.

Suposición 4 – La cantidad de requests recibida en la ventana de tiempo de las estadísticas es manejables

Se asume un flujo de mensajes del orden de 1000 requests por segundo por ejemplo. Con la ventana de tiempo de 1 minuto, esto es una cantidad de requests manejables para el cálculo de estadísticas.

Suposición 5 – Existe un rango horario de baja solicitud del servidor

Entre los requerimientos se menciona un “identificar los mensajes de error con más repeticiones”. Dado que el servidor estará prendido siempre, esta información puede crecer enormemente con el paso del tiempo. En primer lugar, esto implica que se almacenen los datos de manera permanente ya que resultaría imposible mantenerlos en memoria. Y dado que se pide mostrar la información con un cierto orden, será necesario procesar esta información periódicamente. Dado que esta operación podría tardar un tiempo con volúmenes elevados de datos, se propone la definición de un rango horario conocido donde el servidor no recibirá muchos pedidos. Durante este rango horario se actualizara este listado con la información acumulada desde la última actualización.

Suposición 6 – Es despreciable la perdida de mensajes aun en memoria al cerrar la aplicación

En caso de tener que apagar la aplicación por cualquier razón, se considera que no es necesario hacer una descarga de los mensajes recibidos pero aun no procesados o persistidos. Ya que el volumen de

información crece con el paso del tiempo, la perdida de estos pocos datos no resulta significativa para el análisis del funcionamiento del servidor.

Diseño

A partir de los requisitos y las suposiciones planteadas se armó el diseño de la aplicación que se explicará a través de algunos diagramas.

En primer lugar se puede separar la estructura de la aplicación en una primera parte de recibido y extracción de campos de líneas de log, y una segunda que consiste en realizar alguna acción con los campos extraídos. Según los requerimientos existen tres posibles acciones: la actualización de estadísticas temporales, el log a archivos de log totales o de errores, y la actualización de un ranking permanente de errores.

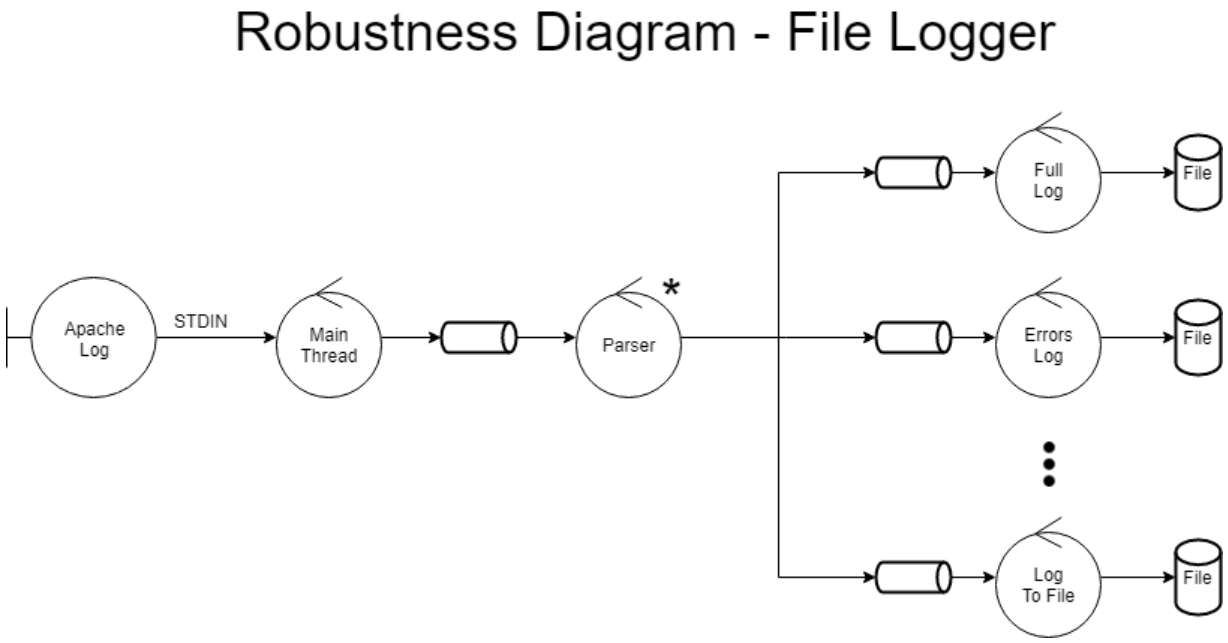


Figura 1: diagrama de robustez, camino de File Logger

Como se ve en la Figura 1 se recibe un mensaje de log desde el servidor Apache que llega a un thread principal que está escuchando mensajes y lo envía a una cola de la que escuchan muchos threads que sirven como Parsers. Estos threads son configurados previamente con pares de expresiones regulares a matchear y colas a donde enviar datos en caso de un match.

En este caso cada par corresponde a un FileLogger. De esta manera se define en la regex tanto que mensajes son importantes para cada FileLogger, como los campos a capturar del mensaje. En caso de match el Parser redirige los campos capturados a través de una cola donde el FileLogger escribe la nueva línea de log a un archivo.

Para configurar los FileLoggers se debe modificar el archivo “config”, en la sección de “config_loggers”. En ese archivo por cada elemento del array se creara una rama de log con una cola, un thread dedicado y un archivo para volcar los logs. Estos threads pasarán la mayor parte del tiempo bloqueados esperando a la cola o al archivo.

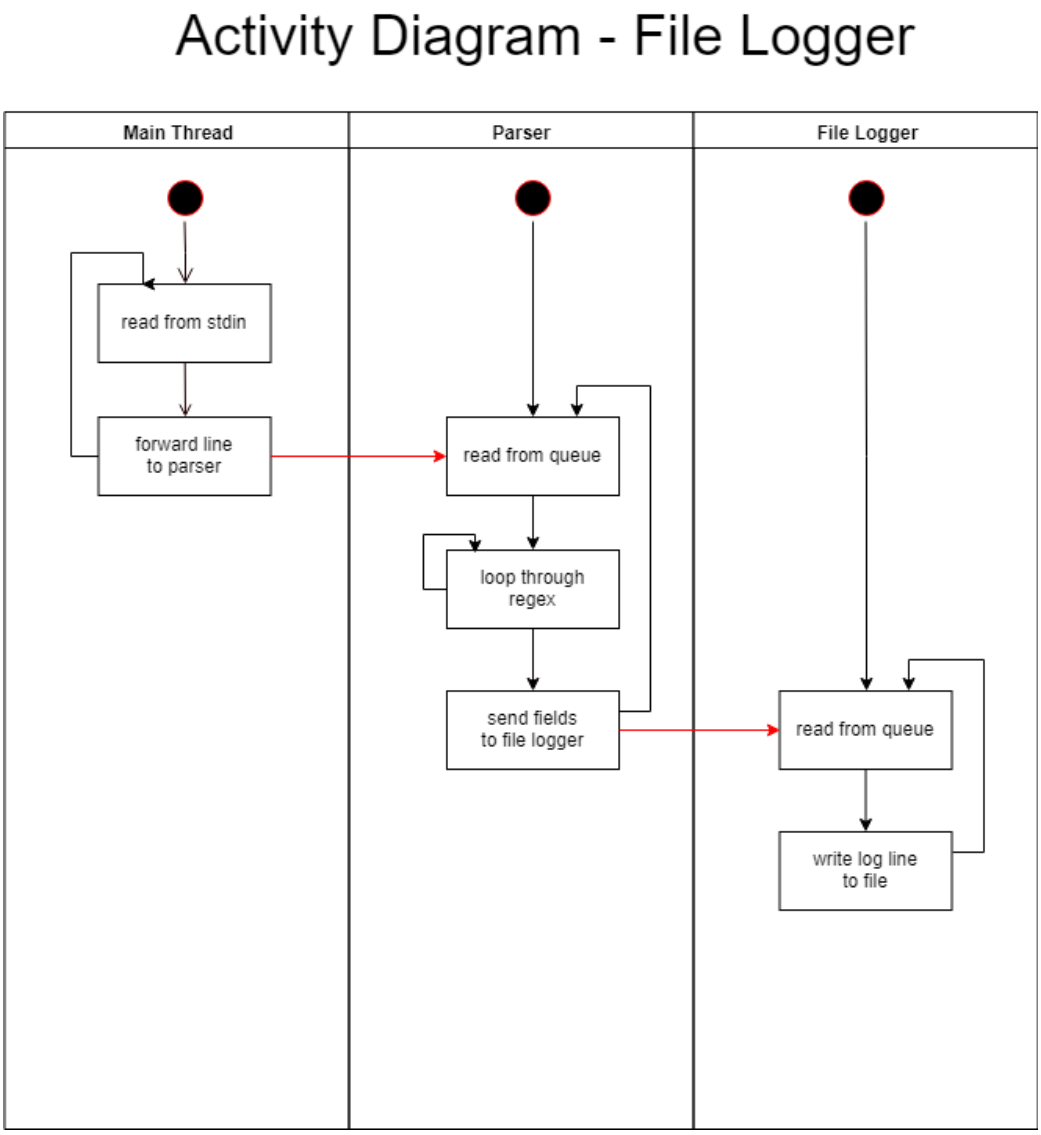


Figura 2: diagrama de actividades, camino de File Logger

En la Figura 2 se puede ver el camino de un mensaje de log desde que es recibido por el thread principal hasta que se guarda a archivo.

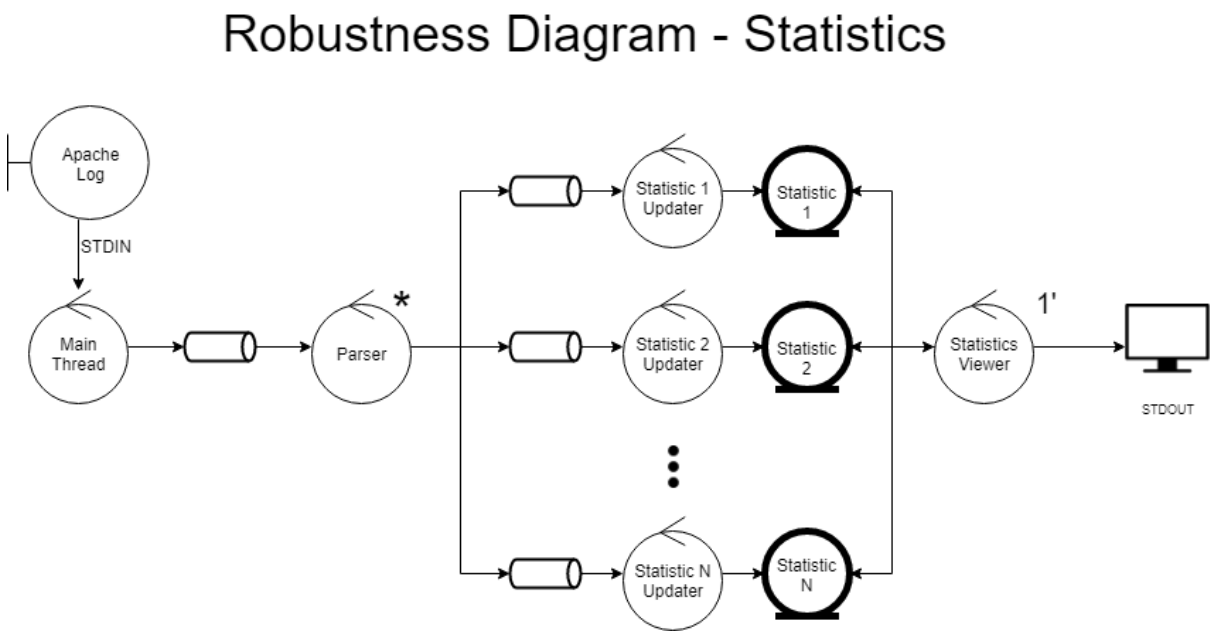


Figura 3: diagrama de robustez, camino de Statistics

Aquí se puede ver el camino de actualización y visualización de estadísticas temporales. La primera parte es igual que antes, hasta llegar al Parser. Luego de esto los mensajes son recibidos por un StatisticUpdater que se bloquea intentando acceder a su Statistic. Este es un objeto compartido por varios threads que cuenta con un Map donde almacena por ejemplo, los distintos clientes y la cantidad de apariciones de cada uno.

Para configurar los valores capturados en las Statistics, es necesario modificar el archivo “config”, en la sección de “config_statistics”. Este archivo cuenta con un elemento por cada rama de StatisticUpdater con su Statistic.

Al final hay un thread llamado StatisticsViewer que se despierta cada un minuto para bloquear las Statistics compartidas, obtener sus valores y limpiarlas a cero. Para no mantenerlas bloqueadas por mucho tiempo se copia en memoria local el Map de cada una. Dado que la ventana de tiempo es pequeña, esto no supone un golpe de performance. Una vez hecho esto, se calculan los valores de cada estadística y se imprimen por pantalla.

Para configurar las estadísticas visibles existe el archivo “config” en la sección de “config_statistics_viewer”. Allí se puede modificar el tiempo que permanece durmiendo este thread, que determina el ancho de la ventana de tiempo para las estadísticas. Además se muestra un posible formato para la configuración y ampliación de las estadísticas sin tener que modificar el código del programa. El parseo de ese formato no fue implementado ya que no parecía importante para el objetivo del trabajo.

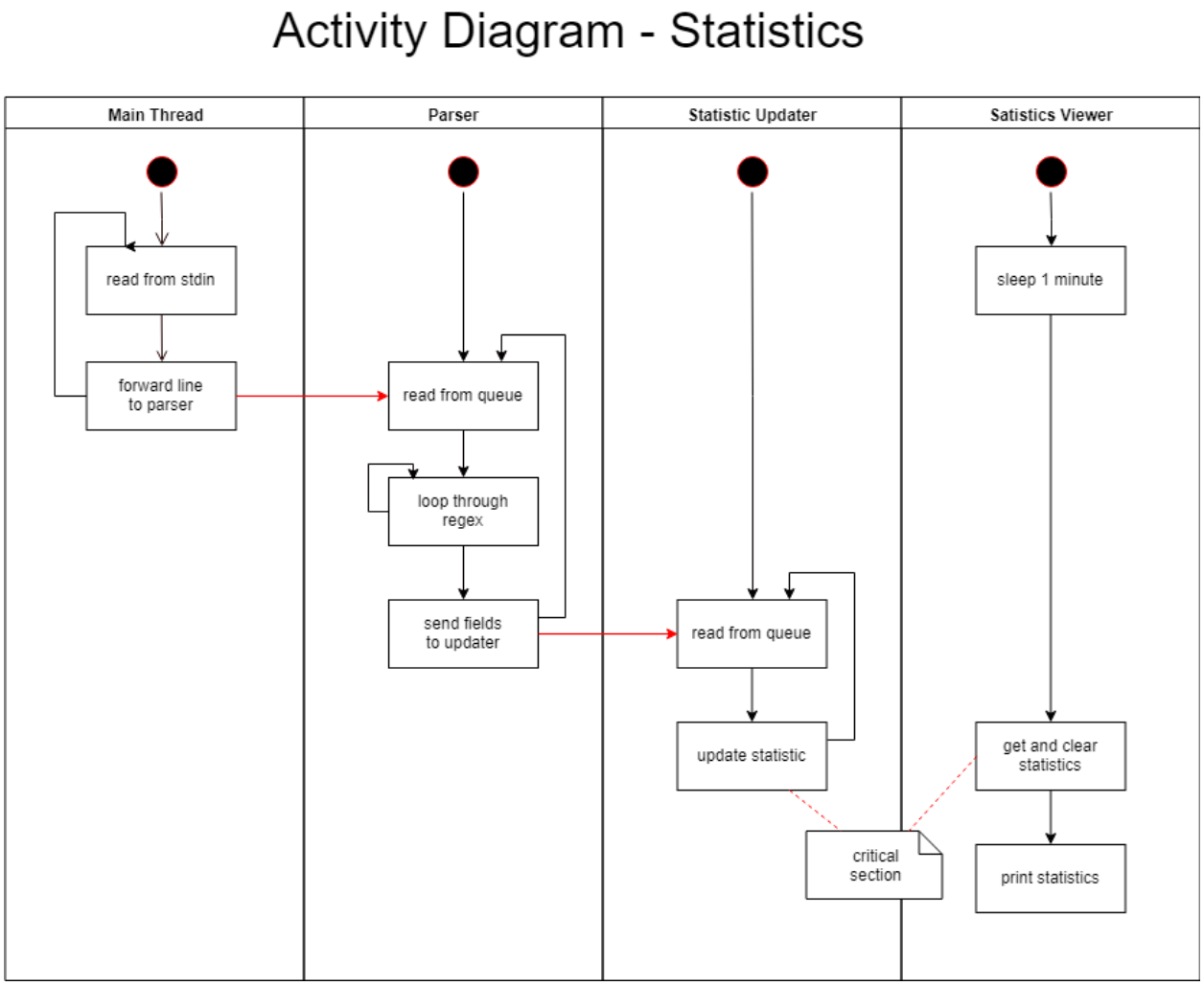


Figura 4: diagrama de actividades, camino de Statistics

En la Figura 4 se puede ver el camino de actualización de una estadística y la acción de mostrarla por pantalla.

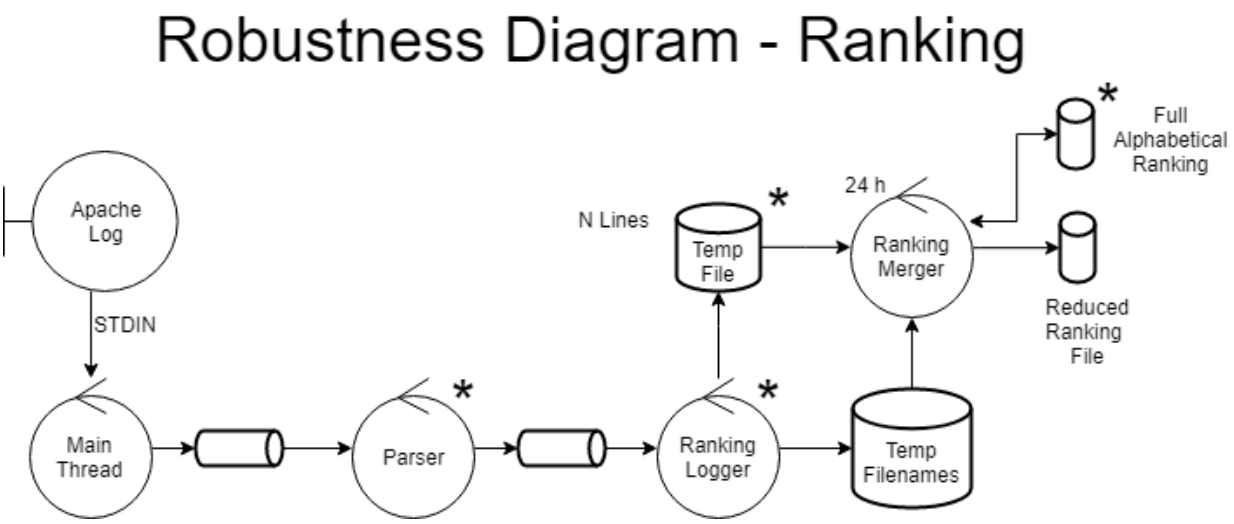


Figura 5: diagrama de robustez, camino de Ranking de errores

Por último está el ranking total de errores más frecuentes. Dado que este ranking total crece infinitamente, no es posible mantenerlo en memoria. La solución propuesta consiste en mantener en un archivo un ranking ordenado y partiendo de la suposición 5 realizar cada cierto tiempo un merge del ranking con los errores recibidos desde la última actualización.

La Figura 5 muestra la estructura de la solución. Existen una serie de threads corriendo como Ranking Loggers que se encargan de guardar archivos temporarios de apariciones de errores. Para esto se acumulan los errores en memoria hasta que hay más de una cantidad N de errores distintos o se almaceno un total de M errores. En ese momento se ordenan y se los guarda a un archivo. Al mismo tiempo se guarda en otro archivo el nombre del archivo temporario generado, para marcarlo como archivo que debe ser procesado. Esto asegura que aun frente a la caída del Monitor, al levantarse nuevamente estos archivos terminados serán procesados. No es así con las apariciones que no hayan sido volcadas a archivo, como explica la suposición 6.

Cada un tiempo determinado (por ejemplo 24 hs) se despierta el thread que corre al Ranking Merger. Este obtiene la lista de archivos de apariciones temporales y la mergea con el archivo más nuevo de Full Alphabetical Ranking para generar uno nuevo. Al mismo tiempo se mantienen en memoria los errores más frecuentes y esto se vuelva a un archivo final de ranking.

Es posible mantener cualquier cantidad de rankings agregando elementos al archivo “config” en la sección de “config_rankings”. Además ahí se puede especificar la cantidad de líneas en cada archivo temporal, la cantidad de threads acumulando apariciones a archivos temporales (por cada ranking), el tiempo de espera entre cada operación de merge y la cantidad de errores a mantener en el archivo ordenado por frecuencia de aparición.

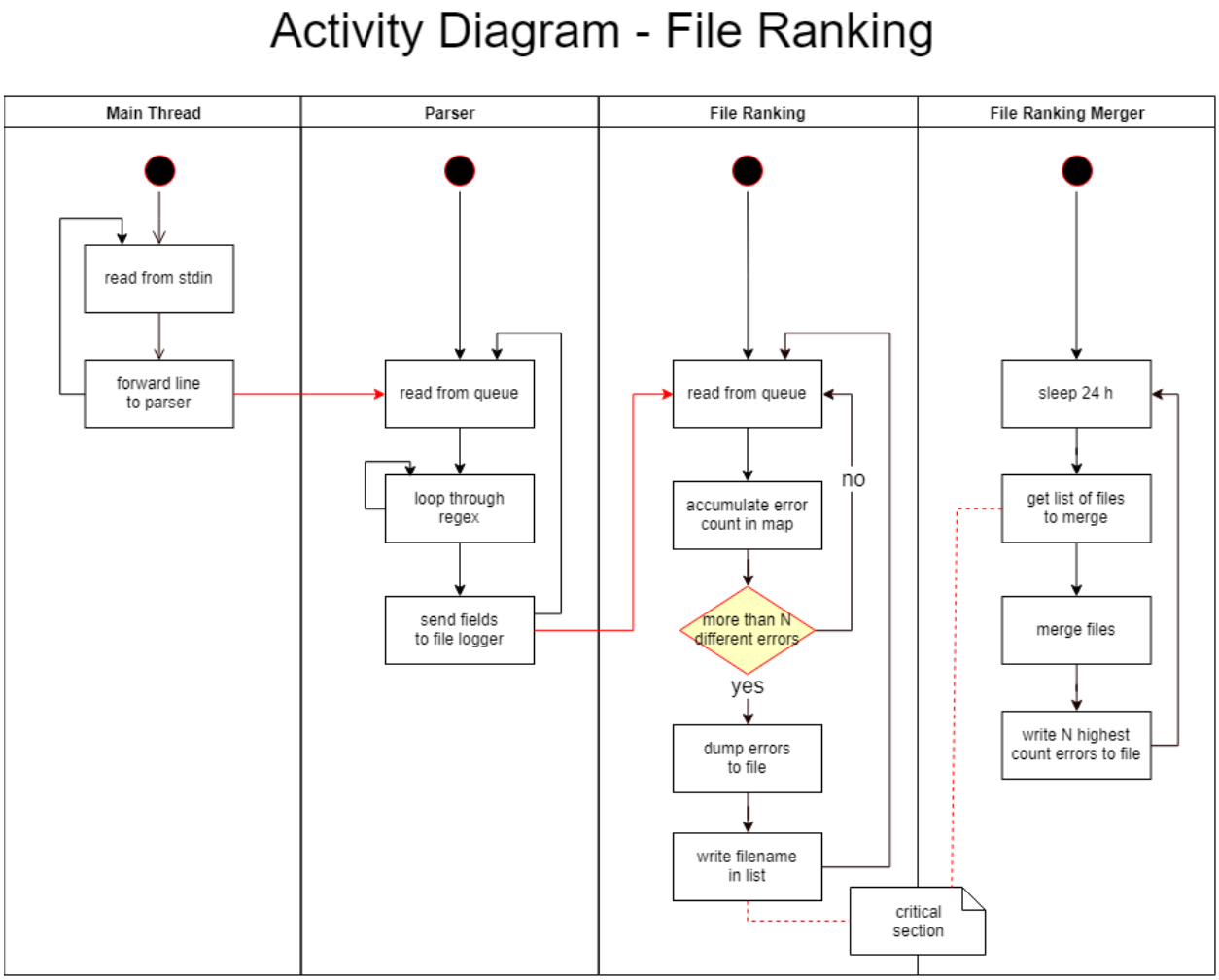
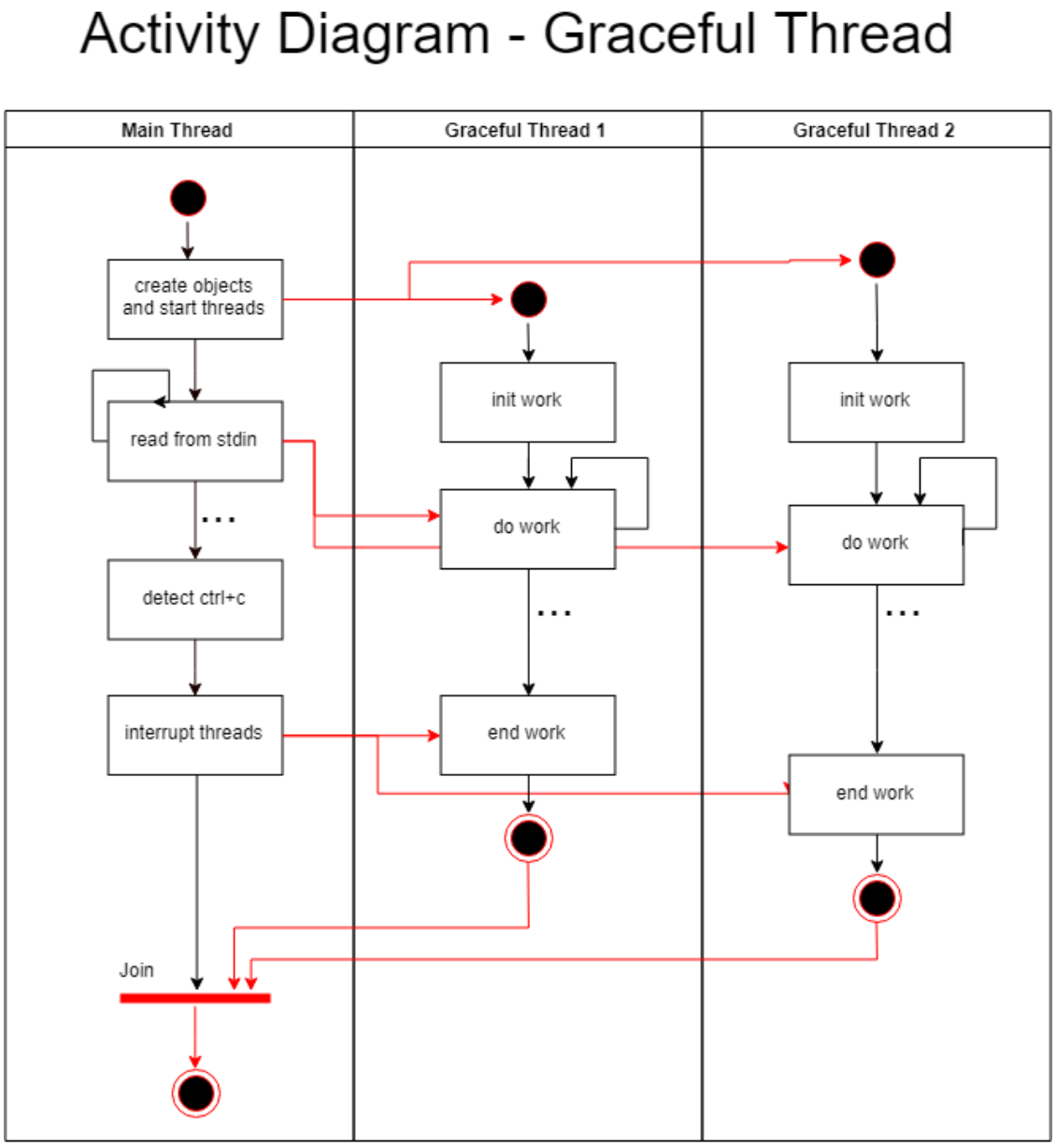


Figura 6: diagrama de actividades, camino de ranking de errores

Con esto quedan cubiertas todas las funcionalidades del Monitor. Falta ver como se inicia y cierra la aplicación.



Como se ve en la Figura 7, se inicial el thread principal que es el que lee los archivos de configuración y crea los objetos y threads acorde. Luego procede a esperar logs y ejecutar los caminos explicados anteriormente. Al detectar un ctrl+c en la entrada se interrumpen todos los threads que se encuentran trabajando, y se procede a esperar a que se cierren todos antes de terminar el programa principal.

Testeo

Para el testeo se generaron archivos de mensajes de log generados al azar según la estructura propuesta. Fueron generados mediante una regex, por lo que no son legibles. De todas maneras es necesario

utilizar alguna herramienta de conteo de un editor de texto dado la cantidad de mensajes, por lo que esto no hace mucha diferencia.

Se incluyen cuatro casos de prueba, con cada vez más mensajes de log para utilizar como entrada de la aplicación. Se pueden encontrar separados en subcarpetas dentro de la carpeta “tests”. Dentro de cada subcarpeta están los archivos de configuración y la entrada para el test, asi como un script para correr el test y otro para limpiar los archivos generados por la corrida.

Test con 100 inputs

Este test está configurado para correr con pocos inputs y mostrar por pantalla los logs completos a medida que los distintos threads se van comunicando entre ellos. Al ser pocos mensajes se puede realizar una inspección manual para ver cómo fueron ocurriendo las cosas. También es posible utilizar alguna herramienta de búsqueda de palabras para verificar si son correctos las estadísticas y el ranking.

Test con 1000 inputs

Es parecido al caso anterior, pero al ser mayor cantidad de logs se muestra solo el output esencial del programa por pantalla. De todas maneras se guarda el output detallado a un archivo que puede ser inspeccionado. El procesamiento de los logs sigue siendo prácticamente instantáneo.

Test con 1000000 inputs

Al ser tantos mensajes de log la ejecución lleva un poco más de tiempo. Esto permite ver en el caso de los rankings como se van generando rankings intermedios a medida que se van procesando los mensajes.

Test con 1000000 inputs para interrumpir con ctrl+c

Este caso es igual al anterior, pero se limita los dumps de rank temporales a un numero bajo de líneas. Esto hace que se creen miles de archivos que deberán ser mergeados, y esta operación tardara notablemente más. De esta forma se puede cerrar la aplicación con ctrl+c en la mitad del procesamiento para verificar que se cierre correctamente. Se puede observar que una vez recibida la señal de salida, el procesamiento frena instantáneamente y además se borra el nuevo rank sin terminar y se restablecen los archivos temporarios para ser procesados la siguiente vez que se arranque el sistema.