Escuela Colombiana de Ingeniería

DIANA LORENA SÁNCHEZ MORENO

Arquitecturas de Software

Introducción al paralelismo - hilos

Trabajo individual o en parejas

Entrega: Martes en el transcurso del día. Entregar: Fuentes y documento PDF con las respuestas.

Parte I Hilos Java

1. De acuerdo con lo revisado en las lecturas, complete las clases CountThread, para que las mismas definan el ciclo de vida de un hilo que imprima por pantalla los números entre A y B.
2. Complete el método main de la clase CountMainThreads para que:
    i. Cree 3 hilos de tipo CountThread, asignándole al primero el intervalo [0..99], al segundo [99..199], y al tercero [200..299].
    ii. Inicie los tres hilos con 'start()'.
    iii. Ejecute y revise la salida por pantalla.
    iv. Cambie el inicio con 'start()' por 'run()'. Cómo cambia la salida?, por qué?.

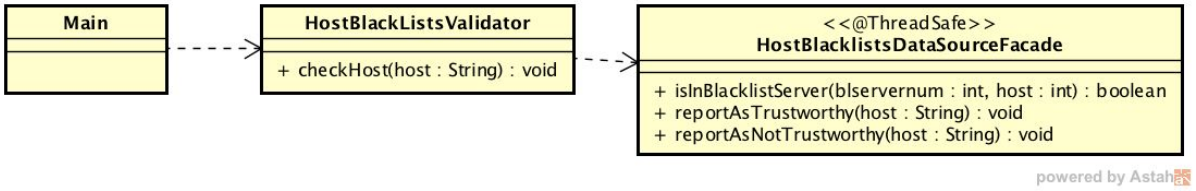
        Con start() se evidencia que los hilos empiezan a ejecutarse al mismo tiempo por lo tanto los números salen en desorden, con el run() los hilos se “turnan” y por eso los números salen ordenados dentro de sus intervalos, por lo que se puede concluir que el start tiene un comportamiento en paralelo y el run un comportamiento en concurrencia.

Parte II Hilos Java

Para un software de vigilancia automática de seguridad informática se está desarrollando un componente encargado de validar las direcciones IP en varios miles de listas negras (de host maliciosos) conocidas, y reportar aquellas que existan en al menos cinco de dichas listas.

Dicho componente está diseñado de acuerdo con el siguiente diagrama, donde:

- HostBlackListsDataSourceFacade es una clase que ofrece una 'fachada' para realizar consultas en cualquiera de las N listas negras registradas (método 'isInBlacklistServer'), y que permite también hacer un reporte a una base de datos local de cuando una dirección IP se considera peligrosa. Esta clase NO ES MODIFICABLE, pero se sabe que es 'Thread-Safe'.
- HostBlackListsValidator es una clase que ofrece el método 'checkHost', el cual, a través de la clase 'HostBlackListDataSourceFacade', válida en cada una de las listas negras un host determinado. En dicho método está considerada la política de que al encontrarse un HOST en al menos cinco listas negras, el mismo será registrado como 'no confiable', o como 'confiable' en caso contrario. Adicionalmente, retornará la lista de los números de las 'listas negras' en donde se encontró registrado el HOST.



Al usarse el módulo, la evidencia de que se hizo el registro como 'confiable' o 'no confiable' se dá por lo mensajes de LOGs:

INFO: HOST 205.24.34.55 Reported as trustworthy

INFO: HOST 205.24.34.55 Reported as NOT trustworthy

Al programa de prueba provisto (Main), le toma sólo algunos segundos analizar y reportar la dirección provista (200.24.34.55), ya que la misma está registrada más de cinco veces en los primeros servidores, por lo que no requiere recorrerlos todos. Sin embargo, hacer la búsqueda en casos donde NO hay reportes, o donde los mismos están dispersos en las miles de listas negras, toma bastante tiempo.

Éste, como cualquier método de búsqueda, puede verse como un problema vergonzosamente paralelo, ya que no existen dependencias entre una partición del problema y otra.

Para 'refactorizar' este código, y hacer que explote la capacidad multi-núcleo de la CPU del equipo, realice lo siguiente:

1. Cree una clase de tipo Thread que representa el ciclo de vida de un hilo que haga la búsqueda de un segmento del conjunto de servidores disponibles. Agregue a dicha clase un método que permita 'preguntarle' a las instancias

del mismo (los hilos) cuantas ocurrencias de servidores maliciosos ha encontrado o encontro.

2. Agregue al método 'checkHost' un parámetro entero N, correspondiente al número de hilos entre los que se va a realizar la búsqueda (recuerde tener en cuenta si N es par o impar!). Modifique el código de este método para que divida el espacio de búsqueda entre las N partes indicadas, y paraleliza la búsqueda a través de N hilos. Haga que dicha función espere hasta que los N hilos terminen de resolver su respectivo sub-problema, agregue las ocurrencias encontradas por cada hilo a la lista que retorna el método, y entonces calcula (sumando el total de ocurrencias encontradas por cada hilo) si el número de ocurrencias es mayor o igual a *BLACK_LIST_ALARM_COUNT*. Si se da este caso, al final se DEBE reportar el host como confiable o no confiable, y mostrar el listado con los números de las listas negras respectivas. Para lograr este comportamiento de 'espera' revise el método join del API de concurrencia de Java. Tenga también en cuenta:
    - Dentro del método checkHost Se debe mantener el LOG que informa, antes de retornar el resultado, el número de listas negras revisadas VS. el número de listas negras total (línea 60). Se debe garantizar que dicha información sea verídica bajo el nuevo esquema de procesamiento en paralelo planteado.
    - Se sabe que el HOST 202.24.34.55 está reportado en listas negras de una forma más dispersa, y que el host 212.24.24.55 NO está en ninguna lista negra.

Parte II.I Para discutir el Martes (NO para implementar aún)

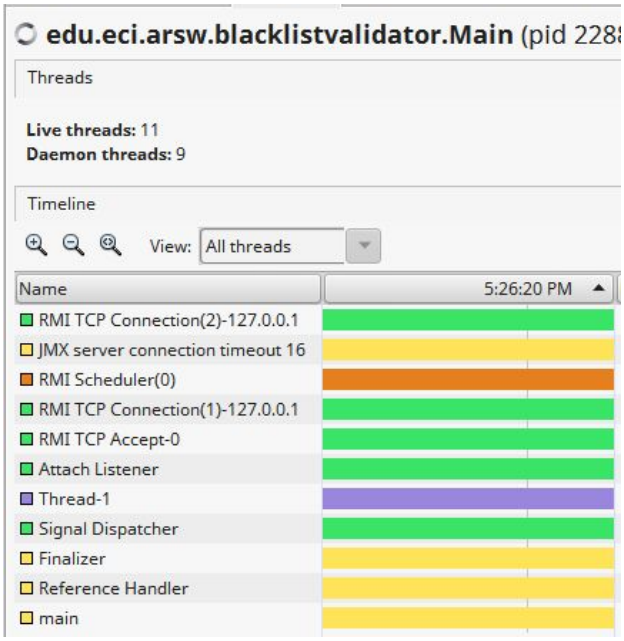
La estrategia de paralelismo antes implementada es ineficiente en ciertos casos, pues la búsqueda se sigue realizando aún cuando los N hilos (en su conjunto) ya hayan encontrado el número mínimo de ocurrencias requeridas para reportar al servidor como malicioso. Cómo se podría modificar la implementación para minimizar el número de consultas en estos casos?, qué elemento nuevo traería esto al problema?

Parte III Evaluación de Desempeño

A partir de lo anterior, implemente la siguiente secuencia de experimentos para realizar las validación de direcciones IP dispersas (por ejemplo 202.24.34.55),

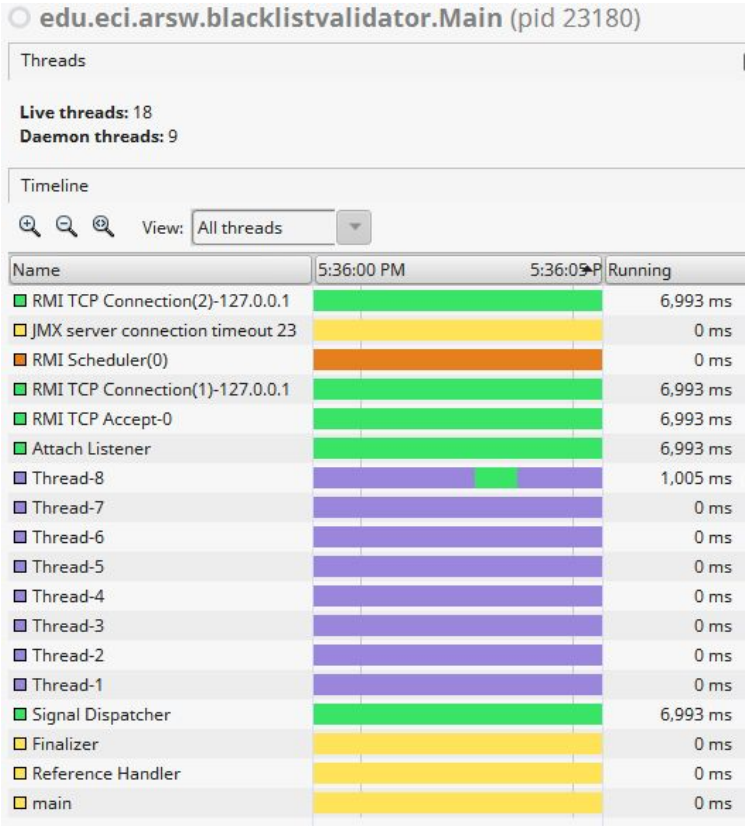
tomando los tiempos de ejecución de los mismos (asegúrese de hacerlos en la misma máquina):

1. Un solo hilo.



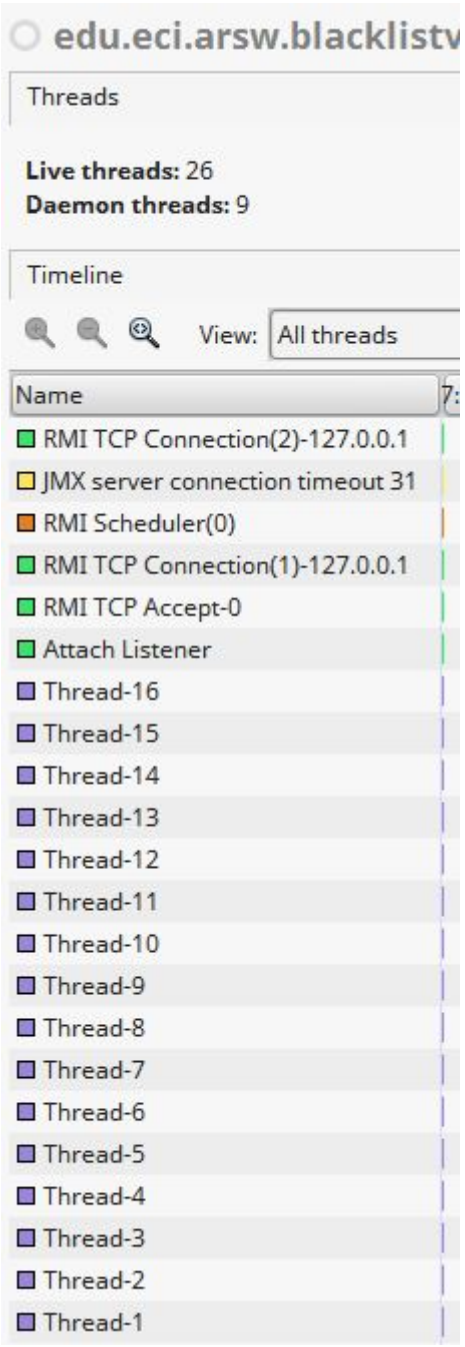

```
INFO: Checked Black Lists:80,000 of 80,000
The host was found in the following blacklists:[29, 10034, 20200, 31000, 70500]
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 1:37.641s
```

2. Tantos hilos como núcleos de procesamiento (haga que el programa determine esto haciendo uso del API Runtime).




```
INFO: Checked Black Lists:80,000 of 80,000
The host was found in the following blacklists:[29, 10034, 20200, 31000, 70500]
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 12.150s
```

3. Tantos hilos como el doble de núcleos de procesamiento.




```
INFO: Checked Black Lists:80,000 of 80,000
The host was found in the following blacklists:[29, 10034, 20200, 31000, 70500]
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 6.325s
```

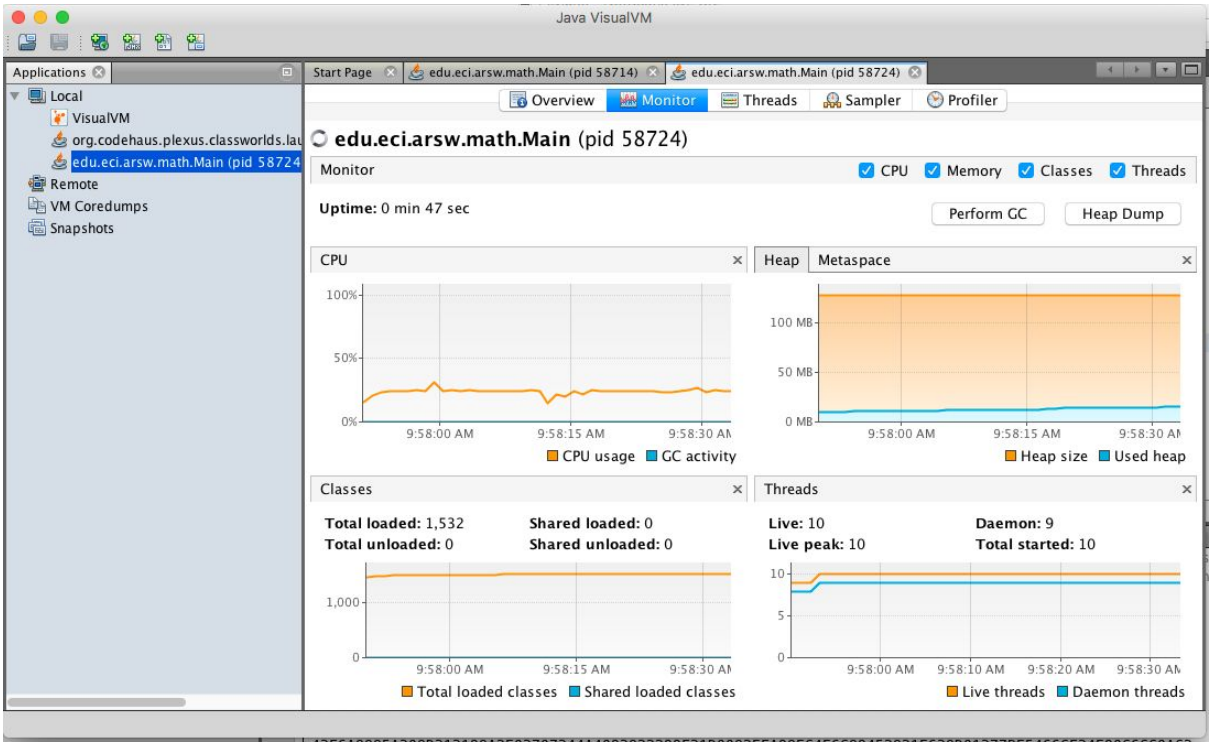
4. 50 hilos.

```
INFO: Checked Black Lists:80,000 of 80,000
The host was found in the following blacklists:[29, 10034, 20200, 31000, 70500]
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 2.147s
```

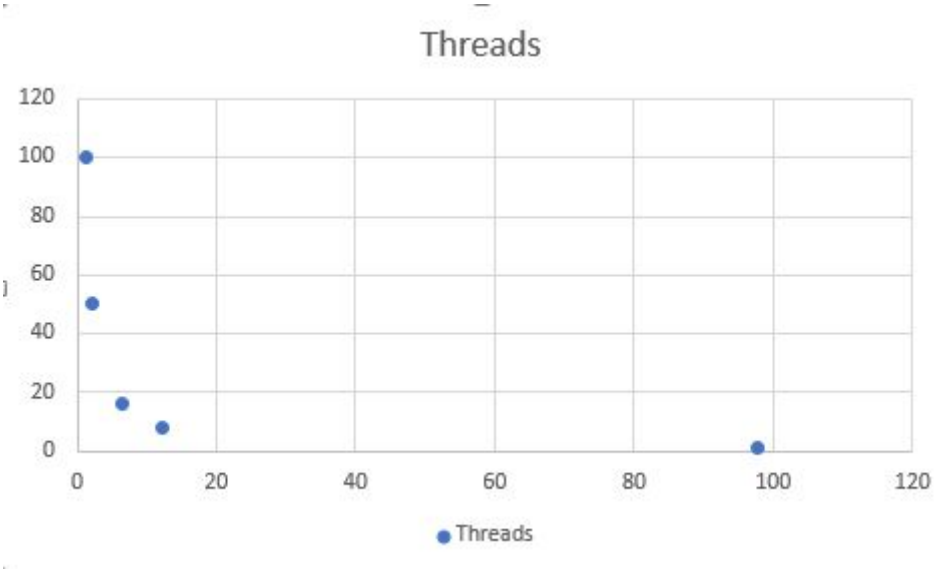
5. 100 hilos.

```
INFO: Checked Black Lists:80,000 of 80,000
The host was found in the following blacklists:[29, 10034, 20200, 31000, 70500]
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 1.321s
```

Al iniciar el programa ejecute el monitor jVisualVM, y a medida que corran las pruebas, revise y anote el consumo de CPU y de memoria en cada caso.



Con lo anterior, y con los tiempos de ejecución dados, haga una gráfica de tiempo de solución vs. número de hilos. Analice y plantee hipótesis con su compañero para las siguientes preguntas (puede tener en cuenta lo reportado por jVisualVM):



1. Según la ley de Amdahls:

$$S(n) = \frac{1}{(1-P) + \frac{P}{n}}$$

, donde *S(n)* es el mejoramiento teórico del desempeño, *P* la fracción paralelizable del algoritmo, y *n* el número de hilos, a mayor *n*, mayor debería ser dicha mejora. Por qué el mejor desempeño no se logra con los 500 hilos?, cómo se compara este desempeño cuando se usan 200?.

   Inicialmente la implementación es más eficiente conforme aumenta el número de hilos, pero cuando los hilos son muchos llega un momento en el que el avance y el tiempo empieza a ser casi constante incluso mayor.

2. Cómo se comporta la solución usando tantos hilos de procesamiento como núcleos comparado con el resultado de usar el doble de éste?.

En ambos casos el tiempo es inversamente proporcional al número de hilos, es decir que cuando se usa el doble de núcleos de procesamiento como hilos, el tiempo es la mitad que cuando se usan el numero de nucleos de procesamiento.

3. De acuerdo con lo anterior, si para este problema en lugar de 100 hilos en una sola CPU se pudiera usar 1 hilo en cada una de 100 máquinas hipotéticas, la ley de Amdahls se aplicaría mejor?. Si en lugar de esto se

usarán c hilos en 100/c máquinas distribuidas (siendo c es el número de núcleos de dichas máquinas), se mejoraría?. Explique su respuesta.

En el caso de 1 hilo en 100 máquinas desaprovecha la capacidad de cada máquina, por lo tanto la mejor opcion seria usar c hilos en 100/c máquinas porque cada una aprovecha su capacidad al máximo.

Criterios de evaluación.

1. Funcionalidad:
    - El problema fue paralelizado (el tiempo de ejecución se reduce y el uso de los núcleos aumenta), y permite parametrizar el número de hilos usados simultáneamente.
2. Diseño:
    - La signatura del método original sólo fue modificada con el parámetro original, y en el mismo debe quedar encapsulado la paralelización e inicio de la solución, y la sincronización de la finalización de la misma.
3. Análisis.
    - Se deja evidencia de la realización de los experimentos.
    - Los análisis realizados son consistentes.