Métricas y desempeño de un servidor/servidor web en Java

Jhusseth Arias, Juan Agustín Lizarazo
Universidad Icesi- Ingeniería de Sistemas
jhussetarias@gmail.com , juan0253@hotmail.com

RESUMEN: Mantener el rendimiento y la calidad es un objetivo importante durante todas las etapas del desarrollo de un producto informático. Las métricas son medidas cuantitativas utilizadas para cumplir con esta meta, dependiendo de su definición los datos y estadísticas arrojadas por los mismos pueden ser utilizadas durante la planificación, organización, control y mejora. La correcta interpretación de esta información permite a los interesados prever y medir el impacto de sus decisiones sobre su producto. En este documento se analizará de forma especifica las métricas IT relacionadas con el desempeño del sistema.

.

I. INTRODUCCIÓN

Las métricas reflejan la realidad de un programa durante todas sus etapas de vida. El análisis de estos datos es en última instancia el factor determinante de las decisiones futuras, por este motivo la elección de las métricas más adecuadas para la evaluación del desempeño de un programa servidor/servidor web en Java es un tema importante a considerar.

Durante el proyecto se toma en consideración inicial la posición de Java frente a otros lenguajes de programación, de esta forma el análisis de los resultados será congruente con la realidad de las circunstancias y se hará una elección correcta de métricas para medir el desempeño de los servidores en Java. Una vez conocidos estos factores ajenos, es posible dar el siguiente paso para encontrar las métricas de hardware adecuadas para el problema.

II. PROBLEMA Y CONTEXTO

A. *Descripción del problema*

Los servidores pueden fallar por causas internas y externas. El correcto monitoreo de su funcionamiento ayuda a evitar estas interrupciones. Las consecuencias de un mal manejo llevan desde la pérdida de información por parte de las compañías, interrupción del servicio de los usuarios e incluso pérdidas monetarias. Un ejemplo claro la noticia [1], que informa en 2013 como la caída de los servidores de Google que representó una pérdida de casi $500.000 dólares y un descenso del 40% de tráfico en internet. Aunque las consecuencias son atenuadas por el manejo de métricas, el lenguaje de programación varía los resultados por factores ajenos a las variables en medición.

B. *Contexto*

La investigación se centrará en el monitoreo interno de los servidores, por este motivo es necesario conocer aspectos importantes sobre el lenguaje de programación a utilizar.

Java es un lenguaje compilado (a un lenguaje intermedio llamado bytecode) e interpretado (por JRE Java Runtime Environment), esta cualidad le permite ejecutarse sin importar el sistema operativo. La elección de un lenguaje de programación tiene un impacto sobre el desempeño del programa como se muestra en la investigación [2], donde se comparan los diferentes lenguajes de programación, según el consumo de memoria, energía y tiempo. Se llega a la conclusión de que no es posible conocer el mejor lenguaje en todos los aspectos, dado a que las diferentes combinaciones de estos factores de investigación arrojaban un resultado distinto.

Sin embargo, se resaltan cualidades de Java (rendimiento, seguridad) y desventajas (Costo, tiempo).

C. *Pregunta de investigación*

¿Cuáles métricas resultan más idóneas para evidenciar el desempeño de Java en un programa de servidor/servidor web?

III. OBJETIVOS

A. *Objetivo General*

Hallar el conjunto de métricas adecuado que permita medir el impacto del tiempo y del costo en un servidor/servidor web Java.

B. *Objetivos específicos*

- Cuando el juego empieza: Encontrar la métrica con mayor impacto temporal en la conexión, reducir el tiempo de conexión en un 20%, presentando modificaciones en los factores de la métrica para atenuar la desventaja temporal de Java
- Durante el primer juego: Encontrar la métrica con mayor impacto en costo, reducir el costo en un 30%, presentando las modificaciones en los factores de la métrica y cambios necesarios en Java.
- Cuando los jugadores se conectan: Establecer las métricas con menor y mayor impacto, establecer los

cambios para una diferencia de 10%, presentando las modificaciones necesarias.

- Durante un juego con múltiples jugadores: Establecer las métricas de mayor impacto temporal, disminuir el promedio por jugador en un 5%, presentando las modificaciones necesarias.
- Cuando los clientes se desconectan: Encontrar la métrica con mayor impacto temporal en la conexión, reducir el tiempo de conexión en un 20%, presentando modificaciones en los factores de la métrica para atenuar la desventaja temporal de Java

IV. ESTADO DEL ARTE

La evaluación del desempeño de un programa con la estructura de servidor/cliente en Java puede considerarse desde diferentes puntos de vista, en este caso se hace especial énfasis en l. Desde el análisis de las excepciones producidas durante el cumplimiento de una funcionalidad como se ve en [3].También el análisis del desempeño general haciendo uso de variadas métricas que finalmente se analizan a partir de gráficos a través de herramientas como CodeMR, hasta el uso de Spring tools[7]. El camino a seguir depende de las restricciones en tiempo para el cumplimiento. Se resume a continuación diferentes soluciones propuestas para medir el desempeño de un programa en Java.

A. Métricas de cobertura de excepciones Def-Catch

Las excepciones en Java responden a condiciones de error, el bloque catch es el punto de partida del código de recuperación para una excepción lanzada en el bloque try. Dentro del artículo[3], se discute sobre la importancia de capturar y manejar las excepciones de forma adecuada, porque estas afectan tanto el funcionamiento como la experiencia del usuario, se especifica el interés en el desempeño de un servidor de Java las excepciones a estudiar serán las pertenecientes a la clase IOException (corresponde a fallas de acceso a disco y red).

Las condiciones necesarias para llevar a cabo esta prueba se encuentran en un análisis inicial. Donde las operaciones con posibilidad de generar una excepción se clasifican en sensibles a fallas y sus asignaciones como vulnerables. Posteriormente, se realiza un análisis sobre la relación existente entre el lugar donde se lanza la excepción y donde se captura. Este evento forma un camino relacionado de varios métodos, a este camino se le llama vínculo E-C. El análisis es asistido por computadora utilizando algoritmos de filtrado DataReach, pero para mayor confiabilidad el tester busca fuentes de error en las conexiones y descarta aquellos que no se consideren E-C.

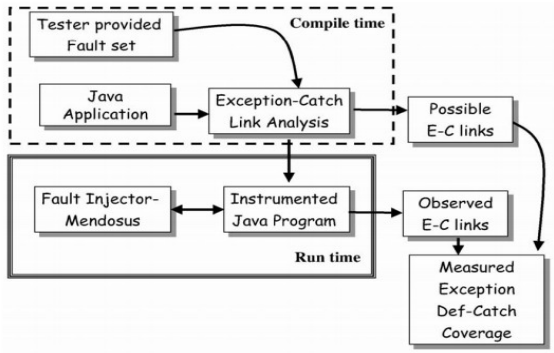

Fig 1. Inyección de errores.

En la figura 1 se muestra el procedimiento para cumplir las condiciones iniciales dentro del tiempo de compilación. En el tiempo de ejecución se observa un inyector de errores, este se encarga de proporcionar a las asignaciones vulnerables valores que lanzarán excepciones. Usando los grafos de llamado (ver figura 2) para el cumplimiento de la funcionalidad de lectura en red y de archivo, se evidencia una ambigüedad en los vínculos E-C, dado a que las asignaciones vulnerables tanto de red o disco dentro de ese contexto generan el mismo camino. Aunque se recomienda ordenar la conexión de estos nodos, realizarlo implica un replanteamiento de muchos otros caminos, lo que se considera costoso en tiempo.

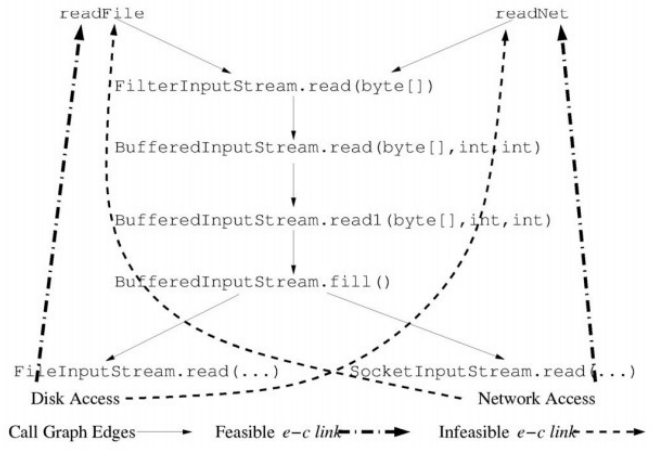

Fig 2. Grafo de llamado

B. Evaluación de métricas con herramienta codeMR

En el documento [4], se define a condeMR como una herramienta de análisis de calidad de Software y código estático, con la capacidad de recopilar datos de un programa, ordenarlos y posteriormente mostrarlos en forma gráfica.

Hace uso de métricas que miden el acoplamiento, la complejidad y el tamaño. Las principales métricas de estudio son LOC, NBD, WMC, RFC, CBO, NST. Dentro del libro [5] kan define estas métricas de la siguiente forma:

Lines Of Code (LOC):

Se realiza un conteo de las líneas de código. Anteriormente cuando la correspondencia de las instrucciones físicas y las instrucciones del programa era de uno a uno, un LOC bajo significaba un buen diseño. Actualmente, no se cuenta con la correspondencia de estas instrucciones y surgen nuevos lenguajes de programación, que obligan a dividir esta métrica en varias subcategorías:

- Conteo de líneas ejecutables,
- Conteo de líneas ejecutables con definición de datos,
- Conteo de líneas ejecutables con definición de datos, y control del lenguaje de trabajo.
- Conteo de líneas físicas en una pantalla de entrada

Nested Blocked Depth (NBD):

Se calcula el nivel de profundidad de una sección de código según las instrucciones anidadas, en Java se identifican gracias a las llaves, una sección de código contenida dentro de 3 llaves se considera de nivel 2, porque las primeras hacen parte del método.

Weighted Methods per Class (WMC):

La complejidad de una clase es definida por la suma de las complejidades de sus métodos, suponiendo a m como una métrica de complejidad. Los datos se obtienen con la siguiente fórmula: $WMC(C) = \sum_{m \in M}$

Response For a Class (RFC):

Esta métrica se encarga de medir la respuesta de una clase, consta de los métodos anidados o llamados directamente por los métodos de la clase en estudio sin importar su visibilidad.

Number of Methods (NOM):

Esta métrica mide el tamaño de una clase tomando como referencia el número de sus métodos. Actúa bajo el supuesto relación directamente proporcional entre cantidad de métodos y tamaño de la Clase.

Coupling Between Objects (CBO):

El acoplamiento se refiere al uso de una clase desde otra. Esta métrica mide el acoplamiento una clase contando las clases acopladas a la misma.

Number of Statements (NST):

Esta métrica mide el tamaño de los métodos con respecto al número de declaraciones que la componen. Estas son independientes al lenguaje de programación, por tanto, es más confiable que la métrica LOC.

C. *Uso de Spring Tools*

Spring

V. METODOLOGÍA

A. *Técnica de recolección de la información*

La recolección de información se llevará a cabo por observación estructurada. Dada la investigación inicial sobre los lenguajes de programación aunado a los trabajos relacionados en métricas sobre los servidores en Java, la hipótesis se define como: Las métricas de medición temporal y de costo tienen los valores menos favorables en un programa servidor/ cliente. El observador no tendrá interacción directa para la consecución de los datos. Sin embargo, puede modificar las condiciones según las necesidades del proyecto. Por este motivo puede considerarse una recolección semiestructurada.

B. *Población y muestra*

La población se encuentra definida por los métodos, atributos y clase del programa servidor. Pese a que resultan afectados los componentes del hardware, estos son el resultado de las interacciones de la población. La muestra se verá definida separando los métodos representativos de la funcionalidad a evaluar, o por defecto haciendo uso del programa de medición.

C. *Técnicas de análisis*

Debido a que los datos se arrojan en gran volumen, resulta beneficioso utilizar la técnica de visualización de datos, de esta forma los patrones se evidencian más fácilmente.

VI. RESULTADOS

VII. CONCLUSIONES

RECONOCIMIENTOS

Poner el reconocimiento a los patrocinadores como una ‘nota al pie’ en la primer página del Trabajo.

REFERENCIAS

[1] «Google se cae durante dos minutos y provoca el descenso del 40% del tráfico mundial», *abc*, 19-ago-2013. [En línea]. Disponible en: https://www.abc.es/tecnologia/redes/20130819/abci-google-trafico-mundial-201308191301.html. [Accedido: 29-nov-2019].

[2] R. Pereira *et al.*, «Energy efficiency across programming languages: how do energy, time, and memory relate?», en *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering - SLE 2017*, Vancouver, BC, Canada, 2017, pp. 256-267.

[3] C. Fu, A. Milanova, B. G. Ryder, y D. G. Wonnacott, «Robustness testing of Java server applications», *IEEE Trans. Softw. Eng.*, vol. 31, n.º 4, pp. 292-311, abr. 2005.
[4] «CodeMR | Documents», *CodeMR*, 04-nov-2018. .
[5] S. H. Kan, *Metrics And Models In Software Quality Engineering*, 2nd Edition. Addison Wesley, 2002.
[7] «Guides». [En línea]. Disponible en: https://spring.io/guides. [Accedido: 29-nov-2019].