Daena: International Journal of Good Conscience. 12(3)423-428. Diciembre 2017.

ISSN 1870-557X

Implementación de patrones de diseño para la optimización de algoritmo de carga masiva en computación en la nube

Implementation of design patterns for the optimization of mass load algorithm in cloud computing

Jesús Ángel Patlán Castillo & Francisco Torres Guerrero

Resumen. La computación en la nube es un servicio que, a pesar de las grandes facilidades que ofrece al desarrollador en el mantenimiento de la aplicación, tiene limitantes en configuraciones que provocan que el sistema web sea programado con algoritmos a medida del servicio de alojamiento. El tipo de algoritmo que se estudia en esta investigación es un algoritmo de carga masiva, el cual es uno de los más exigentes de recursos a nivel computacional. Esta investigación tiene como fin analizar como el uso de patrones de diseño en los algoritmos de carga masiva puede impactar en el rendimiento en el servicio de computación en la nube.

Palabras Claves. Computación en la nube, algoritmo carga masiva, patrones de diseño

Abstract. Cloud computing is a service which, even if it eases the maintenance of the developer's application, has limitations in configurations that cause the web system to be programmed with algorithms restricted by the hosting service. This kind of algorithm studied in the investigation is a bulk-load algorithm, which is one of the most resource demanding at computational level. This investigation has the purpose of analyzing how the use of design patterns in the massive load algorithm can impact in the performance of the cloud computing service.

Key Words. Cloud computing, massive load algorithm, design patterns

Introducción

En la última década, la computación en la nube ha sido una forma muy eficaz de trabajar para los desarrolladores de sistemas web, así como también de facilitar al usuario final de la utilización de un servidor en línea con características que se adecuan a sus necesidades. Sin embargo, en ocasiones este tipo de servicios no permiten que el usuario pueda configurar cierto tipo de aspectos del servidor que limitan la capacidad del mismo. Es aquí donde los patrones de diseño pueden dar la diferencia.

Los patrones de diseño empezaron a estudiarse a partir de la década de los 90 por un grupo de desarrolladores conocidos como “The Gang of four”, conformado por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Con ellas, se tenía la intención de establecer “normas” que pudieran ayudar a crear un código más eficaz y eficiente para la comunidad de programadores. Con el tiempo, fueron agregándose más patrones de diseño con los cuales permitían mejorar aun más el rendimiento de las aplicaciones.

Dado que la computación en la nube cuenta con servicios de prestaciones de servidores en la nube, en este documento se realizará una prueba en la que observaremos que impacto tiene la utilización de los patrones de diseño en este tipo de servicio, con el fin de comprobar que la mejora de rendimiento pueda adecuarse a las limitantes que tienen los servidores en línea.

Marco teórico

Computación en la nube: Definido por la NIST (National Institute of Standards and Technology) como “un modelo que permite ubicuidad, conveniencia y acceso de red en demanda a un conjunto de recursos computacionales que pueden ser rápidamente suministrado y liberado con un esfuerzo mínimo de administración o de interacción con el proveedor”.

Patrones de diseño: Definido por “The Gang of four” como “una forma de reusar conocimiento abstracto acerca de un problema y su solución”. Existen diversos patrones de diseño, así como también anti patrones de diseño, entre los que se utilizarán en esta investigación son:

- Patrones de diseño
    - Object Pool: Trata de dar preferencia a la clonación de objetos en lugar de la creación de nuevos, con el fin de disminuir la carga computacional que implica crear objetos complejos.

  - Flyweight: Su fin es minimizar la cantidad de objetos si existen varios que poseen una información idéntica.
- Anti patrones de diseño
  - Accidental Complexity: Es utilizar una función con complejidad innecesaria.
  - Lava Flow: Tener código sin ningún uso en el sistema.
  - Packratting: Mantener objetos vivos que provocan el consumo de memoria del servidor.

Heroku: Es una plataforma que permite el alojamiento de servicios web en la nube.

PHP (Preprocesador de Hipertexto): Lenguaje de programación encargado de compilar código de páginas web del lado del servidor.

Estado del arte

1. Mansoor, U., Kessentini, M., Maxim, B. R., & Deb, K. (2017). Multi-objective code-smells detection using good and bad design examples. *Software Quality Journal*, *25*(2), 529-552.
2. Yin, H., Zou, T., & Xie, H. (2017). *U.S. Patent No. 9,729,424*. Washington, DC: U.S. Patent and Trademark Office.
3. Chase, J., & Niyato, D. (2017). Joint optimization of resource provisioning in cloud computing. *IEEE Transactions on Services Computing*, *10*(3), 396-409.
4. Matczynski, M. J., Curtis, P. M., & Kellett, O. F. (2017). *U.S. Patent No. 9,584,435*. Washington, DC: U.S. Patent and Trademark Office.

Experimentación

Para la realización de nuestra investigación, utilizamos el servicio que ofrece la plataforma Heroku de un servidor para suministrar una aplicación web codificada en PHP 7.2.1, el cual cuenta con las siguientes características:

Características del Servidor

Almacenamiento	- 500MB para la aplicación - Hasta 10,000,000 de registros en la base de datos
Memoria RAM	512MB

La aplicación web trata de un generador de registros de actividades, los cuales se crean en base a una combinación entre los siguientes campos:

- Una fecha
- Un usuario
- Una ubicación
- Un producto

Teniendo un algoritmo de carga masiva, la aplicación generará los registros y después los registrará en la base de datos. Una vez realizado esto, se terminará el proceso y se realizarán las mediciones utilizando el tiempo de respuesta que se tiene desde que se envía la petición hasta que se recibe la respuesta por parte del servidor.

Los patrones de diseño que utilizaremos son object pool y flyweight, y los anti patrones de diseño a utilizar son accidental complexity, lava flow, packratting. Se hará una combinación en la utilización de estos 5 patrones en el código de la aplicación.

Resultados

Se hicieron distintas combinaciones con los patrones de diseño y se generaron distintas cantidades de registros. Se hicieron 10 pruebas con cada número de registros distintos y se tomo el promedio del tiempo de respuesta que llego del servidor:

UTILIZANDO LOS PATRONES: ACCIDENTAL COMPLEXITY, LAVA FLOW, PACKRATTING	
NÚMERO DE REGISTROS	Tiempo de respuesta
50,000	10.23 segundos
100,000	16.56 segundos
150,000	24.76 segundos
200,000	29.21 segundos

UTILIZANDO LOS PATRONES: PACKRATTING	
NÚMERO DE REGISTROS	Tiempo de respuesta
50,000	9.57 segundos
100,000	15.41 segundos
150,000	22.98 segundos
200,000	28.45 segundos

UTILIZANDO LOS PATRONES: ACCIDENTAL COMPLEXITY, LAVA FLOW	
NÚMERO DE REGISTROS	Tiempo de respuesta
50,000	8.59 segundos
100,000	13.98 segundos
150,000	21.25 segundos
200,000	27.34 segundos

UTILIZANDO LOS PATRONES: OBJECT POOL, FLYWEIGHT	
NÚMERO DE REGISTROS	Tiempo de respuesta
50,000	7.09 segundos
100,000	11.89 segundos
150,000	19.58 segundos
200,000	25.39 segundos

Conclusiones

Comparando los resultados obtenidos en las diversas pruebas, podemos notar que el tiempo de respuesta fue disminuyendo a medida que fueron implementados los diseños de patrones en el código del sistema y también aumentaba el rendimiento conforme fueron resolviéndose los anti patrones de diseño que se utilizaron dentro del código. Se puede observar que hay aproximadamente un 30% de optimización en el rendimiento del servidor utilizando un código que cumple con los patrones de diseño propuestos por “The Gang of four”.

Por tanto, en base a los resultados obtenidos, podemos asegurar que el utilizar un código que acate los patrones de diseño de software que se han propuesto impactan en el rendimiento de un servidor de ofrecido por un servicio de computación en la nube.

Referencias

1. Mell, P., & Grance, T. (2011). The NIST definition of cloud computing.
2. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design patterns: elements of. 1994.
3. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., ... & Zaharia, M. (2009). *Above the clouds: A berkeley view of cloud computing* (Vol. 4, pp. 506-522). Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.
4. Wolfgang, P. (1994). Design patterns for object-oriented software development. *Reading Mass*, 15.
5. Duyne, D. K. V., Landay, J., & Hong, J. I. (2002). *The design of sites: patterns, principles, and processes for crafting a customer-centered Web experience*. Addison-Wesley Longman Publishing Co., Inc..
6. Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, *25*(6), 599-616.
7. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1993, July). Design patterns: Abstraction and reuse of object-oriented design. In *European Conference on Object-Oriented Programming* (pp. 406-431). Springer, Berlin, Heidelberg.
8. Martin, R. C. (2002). *Agile software development: principles, patterns, and practices*. Prentice Hall.
9. Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). *Head First Design Patterns: A Brain-Friendly Guide*. " O'Reilly Media, Inc.".
10. Erl, T., Cope, R., & Naserpour, A. (2015). *Cloud computing design patterns*. Prentice Hall Press.
11. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... & Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM*, *53*(4), 50-58.