Evaluación y análisis de rendimiento de los frameworks de persistencia Hibernate y Eclipselink*[1]

[Assessment and analysis of performance persistence frameworks Hibernate and EclipseLink]

Mauro CALLEJAS CUERVO[2], Diego Iván PEÑALOSA PARRA[3]
Andrea Catherine ALARCÓN ALDANA[4]

Recibo: 20.02.2011 - Ajuste: 14.03.2011 - Ajuste: 24.04.2011 - Aprobación: 30.04.2011

Resumen

Se recopila el estudio delimitado y particular de la evaluación de rendimiento realizada a los framework de persistencia Hibernate y EclipseLink, presentando una fundamentación teórica, que permitió desarrollar el objeto de estudio de esta investigación de forma analítica, crítica y primordialmente buscando que los resultados obtenidos e ilustrados en este trabajo sirvan de aporte

* Modelo para citación de este artículo científico
CALLEJAS CUERVO, Mauro; PEÑALOSA PARRA, Diego Iván y ALARCÓN ALDANA, Andrea Catherine (2011). Evaluación y análisis de rendimiento de los frameworks de persistencia Hibernate y Eclipselink. En: Ventana Informática. No. 24 (ene-jun., 2011). Manizales (Colombia): Universidad de Manizales. p. 9-23. ISSN: 0123-9678

1 Artículo proveniente del proyecto *Evaluación de rendimiento entre los frameworks de persistencia Hibernate y Eclipselink*, ejecutado en el periodo 10/12/2009 –24/05/2010, e inscrito en el Grupo de Investigación en Software GIS de la UPTC. Realizado como opción de grado para obtener el título de Ingeniero de Sistemas y Computación, dirigido por Mauro Callejas Cuervo.

2 Ingeniero de Sistemas, Especialista en Ingeniería de Software, Magister en Ciencias Computacionales. Estudiante de Doctorado en Ciencia y Tecnología Informática.
Profesor Asistente, Universidad Pedagógica y Tecnológica de Colombia-UPTC. Director del Grupo de Investigación en Software-GIS de la UPTC. Tunja (Colombia). Correo electrónico: maurocallejas@gmail.com.

3 Ingeniero de Sistemas y Computación.
Investigador Grupo de Investigación en Software GIS, Universidad Pedagógica y Tecnológica de Colombia,Tunja (Colombia). Correo electrónico: diegopenalosa@gmail.com.

4 Ingeniera de Sistemas y Computación, Especialista en Ingeniería de Software, Magister en Software Libre de la Universidad Oberta de Cataluña de España-UNAB.
Profesora UPTC, Facultad de Ingeniería, Escuela de Sistemas y Computación. Investigadora Principal del Grupo de Investigación en Software GIS de la UPTC.Tunja (Colombia). Correo electrónico:acalarcon@gmail.com.

en el momento de tomar la decisión de cuál de los dos presenta mayor rendimiento. Además se muestran los resultados obtenidos en cada una de las pruebas de estrés que se implementaron para la respectiva valoración, basadas en inserción, actualización, borrado y consulta de un número significativo de registros.

Palabras Clave*: EclipseLink,Hibernate, Marco de trabajo, Rendimiento*

Abstract

It compiles the study defined and particular performance evaluation conducted to Hibernate persistence framework and EclipseLink, presenting a theoretical basis, allowed to develop for the purpose of this research an analytical, critical and primarily looking to the results and illustrated in this work serve as input when making the decision on which of the two has better performance. Also depicted are the results of each of the stress tests were carried out for the respective assessment, based on insert, update, delete, and query for a significant number of records.

Keywords*: EclipseLink, Hibernate, Framework, Performance*

Introducción

En el proceso de desarrollo de software un aspecto vital es, el correcto y eficiente funcionamiento de la base de datos y por supuesto el acceso a esta; de igual forma como existen *framework*s para desarrollo de aplicaciones (Vasa et al., 2007) también los hay para manejo de persistencia (Hemrajani, 2006), los cuales adquieren cada vez mayor importancia haciendo necesario su uso, por tanto es necesario tener claridad en los aspectos que se deben evaluar en el momento de seleccionar uno de ellos.

En la generación de aplicaciones informáticas, específicamente usando herramientas como Java (Gosling et al., 2005), surge la incógnita sobre ¿Cuál de los *framework* de persistencia brinda mayor rendimiento?, la documentación existente para dar una respuesta objetiva a esta pregunta es mínima, convirtiendo esta decisión en un problema común para los desarrolladores.

Esta investigación aborda conceptos relacionados con el uso y evaluación de *framework*s de persistencia, enfatizando en el estudio de rendimiento de Hibernate y EclipseLink, además de la instalación del software requerido para la evaluación propuesta, ejecución de pruebas y conclusiones basadas en el método y herramientas definidas.

La estructura del artículo presenta en primera instancia la fundamentación teórica de la investigación, posteriormente se hace una descripción breve de la metodología que permitió el desarrollo del análisis, luego se presentan los resultados y se hace una discusión sobre lo obtenido, para finalmente presentar las conclusiones y referencias consultadas.

1. Fundamentación teórica

Actualmente, existen *framework*s para la mayoría de los lenguajes de programación, su definición varía, pero la mayoría apunta a exponer que es una estructura de soporte con características genéricas, el cual puede ser utilizado en un proyecto de software de iguales características técnicas; los que mayor auge han tenido en los últimos años son los destinados al manejo de persistencia basados en ORM (*Object Relational Mapping*) (O'Neil, 2008), convirtiéndose en una práctica necesaria en la manipulación de objetos entre la memoria y su ubicación real en la base de datos (Wei et al., 2009), (Zyl et al., 2009), permitiendo al desarrollador definir de forma clara la gestión entre el modelo de objetos y el esquema de base de datos, además de expresar las operaciones de esta última en términos de objetos (Richardson, 2009). Estos *framework*s buscan simplificar el proceso de desarrollo (Benjelloun, 2006). Para poder realizar una comparación objetiva entre *Hibernate* y *EclipseLink*, es necesario antes, tener claros conceptos como ORM, JPA y *Framework* de persistencia, para no generar confusiones al respecto, es así que antes de abarcar los conceptos principales del tema a tratar se exponen algunas definiciones relevantes.

1.1 Object Relational Mapping (ORM)

Traduce Mapeo objeto-relacional. Esta tecnología, utilizada en el proceso actual del desarrollo de software, pretende automatizar el puente entre el mundo orientado a objetos y el mundo relacional, reduce la duplicación de datos, el costo de mantenimiento y la susceptibilidad a errores asociados a ella. Hoy en día, existe variedad de herramientas ORM y otras están en desarrollo; sin embargo, algunas de las existentes no cumplen con el objetivo de la persistencia transparente de datos, y por esto no obtienen popularidad en la industria del software (Barnes, 2007).

1.2 Java Persistence API (JPA)

Proporciona un modelo de persistencia POJO (*Plain Old Java Object*) de mapeo objeto-relacional. «*Es la interfaz de programación de aplicaciones (API) de persistencia y del estándar ORM de la plataforma Java EE 5.0, nació como parte del esfuerzo del grupo de expertos del*

estandar EJB 3.0 (Enterprise Java Beans), pero su uso no se limita a los componentes de software de EJB, también puede ser utilizado directamente por las aplicaciones web y clientes de aplicaciones, e incluso fuera de la plataforma Java EE, por ejemplo, en aplicaciones Java SE» (Sun, 2010).

1.3 Framework

La definición utilizada con frecuencia para *framework*, es la propuesta por Johnson y Foote (1988): «*un framework es una aplicación reutilizable, semi-completa que puede ser especializada para producir aplicaciones concretas y específicas. El framework describe los objetos que componen el sistema, cómo éstos interactúan, y cuáles son sus responsabilidades*».

1.4 Framework de persistencia

Un *Framework* de persistencia simplifica el proceso de desarrollo de software, ya que mueve los datos del programa en su forma más natural (objetos en memoria), a la base de datos. Es así que gestiona la base de datos y el mapeo entre ésta y los objetos (Benjelloun, 2006).

1.5 Garbage collector

Es una forma de gestión de memoria automática. El recolector de basura libera la memoria que ha sido ocupada por los objetos dentro de la aplicación y que posteriormente no tendrán ningún uso. La recolección de basura fue inventada por John McCarthy en el año 1959, para resolver los problemas de la gestión manual de memoria en el lenguaje de programación Lisp. (Sun, 2008).

1.6 Rendimiento

En términos generales de sistemas de información, el rendimiento es considerado como la «*reducción de uso de la CPU y la búsqueda de métodos más óptimos para el sistema*» (Nolte, 1995). De igual manera es necesario tener en cuenta aspectos como *throughput* o volumen de trabajo o de información que fluye a través del sistema, así como el tiempo de respuesta, el costo por transacción y medidas de utilización de recursos, para evaluar el rendimiento de una aplicación (Throughput, 2010).

1.7 Pruebas de rendimiento

Para llevar a cabo la medición y evaluación del rendimiento, están definidas distintas pruebas que en general pueden considerarse como lo indica Tuya, et al. (2007, 43), que afirman que las pruebas para determinar si los tiempos de respuesta del sistema tanto en condiciones normales como en condiciones especiales, se encuentran dentro de

los limites predefinidos; de forma similar Alonso (2005), afirma que las pruebas tienen como propósito evaluar el tiempo de repuesta del sistema o memoria que ocupa.

Por otro lado, Pressman (2010), argumenta que es inaceptable que un software proporcione las funciones requeridas pero no se ajuste a los requisitos de rendimiento, por esto la importancia de estas pruebas diseñadas para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado.

1.8 Prueba de estrés o esfuerzo (Stress Testing)

Este tipo de pruebas se enfocan en la determinación o validez de las características de rendimiento del sistema cuando se someten a condiciones extremas durante las operaciones de producción (Alonso, 2005).

1.9 Hibernate

Es una herramienta de persistencia objeto – relacional y de consulta, que permite desarrollar clases persistentes siguiendo el lenguaje orientado a objetos, incluyendo la asociación, herencia, polimorfismo, composición y colecciones; además proporciona un lenguaje de consulta orientado a objetos llamado *HQL* (*Hibernate Query Language*), similar a SQL, con un criterio orientado a objetos (Hibernate, 2011).

La evolución y mejoramiento de *Hibernate*, no sólo cuenta con el grupo de desarrollo sino también con sus usuarios, ya que los foros (Hibernate, 2009) se convierten en instrumentos de evaluación que permiten precisar posibles errores o defectos en el desarrollo de este. El grupo de desarrollo sigue trabajando con el ánimo de lograr como meta aliviar en un 95% el trabajo del desarrollador en el manejo de la persistencia de datos (King et al., 2011).

1.10 EclipseLink

Eclipse Persistence Services Project (EclipseLink), ofrece una amplia solución de código abierto para la gestión de persistencia en Java, ofreciendo alto rendimiento y escalabilidad para desarrolladores de software empresarial, utilizando fuentes de datos, formatos y contenedores (EclipseLink, 2011).Éste *framework* apoya una serie de estándares de gestión de persistencia, tales como JPA y ORM nativo, con JAXB (API Java para XML Binding) y SDO (Service Data Objects).

EclipseLink es considerado como el *framework* sucesor de *TopLink* (Oracle, 2011), pues está basado en el código fuente y las pruebas realizadas por *Oracle* a este proyecto antes de donarlo a la fundación de *Eclipse*.

1.11 Comparación de frameworks

En la preocupación de la selección correcta del uso de un determinado *framework* de persistencia, previamente se han realizados distintas comparaciones entre estos. Los aspectos comúnmente evaluados son los que tienen que ver con cual tiene un mayor rendimiento en cuanto a uso memoria, tiempo de ejecución, entre otros aspectos (Zyl et al. (2006); Goldschmidt et al., (2008); Kalantari y Bryant (2009); White, (2010) y Acharya (2007)).

2. Metodología

Para llevar a cabo la evaluación de los *framework*s en cuestión, a continuación se describen de manera breve las actividades tenidas en cuenta para la comparación y posteriorpresentación de los resultados.

2.1 Aspectos seleccionados para evaluar

Después de haber revisado las prácticas más comunes de evaluación en cuanto a *framework* de persistencia se refiere y con base en casos propios se decidió que las métricas de rendimiento son las expuestas en la tabla 1.

Tabla 1. Métricas y unidades de medida seleccionadas para la evaluación.

Métrica	Unidad
Tiempo empleado en la ejecución de determinadas instrucciones	Milisegundos
Memoria RAM consumida en la ejecución de algunas instrucciones indicadas	Megabyte
Uso de GarbageCollector	Porcentaje

Para cada uno de estos indicadores se presentará un gráfico de tiempo en comparación con el número de operaciones realizadas con éxito.

2.2 Descripción de la aplicación desarrollada

Para llevar a cabo el proceso de pruebas se desarrolló una aplicación en java que ejecutará automáticamente las instrucciones SQL apropiadas para la investigación.

La aplicación desarrollada consta de tres paquetes, como se expone en la figura 1. En el paquete lógica se encuentran las clases y archivos de mapeo propios de cada *framework*, en el paquete denominado db

se encuentra la clase sesión y persistencia encargadas de interactuar con las clases del *framework* evaluado en un momento determinado, adicionalmente dentro del paquete db se encuentra una clase propia de cada *framework* que es la encargada de crear las conexiones con cada uno de ellos. En el paquete test se encuentra la interfaz de usuario para ejecutar cada una de las pruebas a realizar en la evaluación.

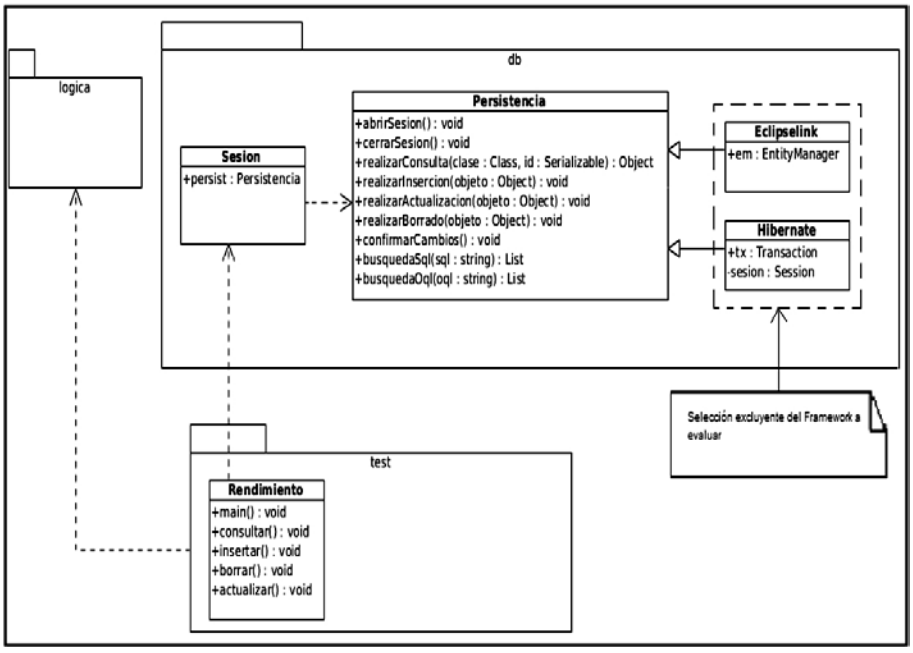

Figura 1. Diagrama de Clases de la aplicación diseñada.

La figura 2 presenta la interfaz de la aplicación desarrollada, en la cual se observa que la funcionalidad única y principal de ésta es permitir la ejecución de los comandos SQL usados para las pruebas de rendimiento, y que se describen posteriormente.

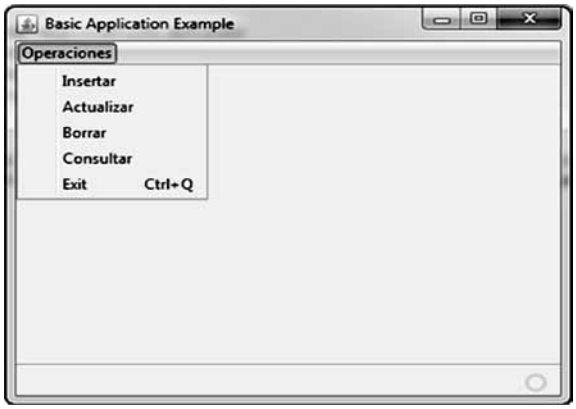

Figura 2. Interfaz de usuario de la aplicación

2.3 Pruebas seleccionadas para el proyecto

El tipo de pruebas que se llevaron a cabo en esta evaluación fueron las pruebas de estrés o esfuerzo, las cuales tienen como objetivo medir el rendimiento bajo cargas elevadas de trabajo (Aycart et al., 2007), concretamente se forzó a que los *framework* ejecutaran grandes cantidades de instrucciones SQL continuas.

2.3.1 Aspectos técnicos de las pruebas.Los requerimientos técnicos del equipo de pruebas utilizadas en esta investigación se exponen en la tabla 2.

Tabla 2. Requerimientos técnicos del equipo de pruebas.

Hardware	Software
Computador portátil Toshiba Satellite L305D-S5893, procesador AMD Turion X2 TL60 2,0 GHz, RAM de 4 Gb DDR2	Sistema operativo Windows 7 Ultimate de 64 Bits, se ejecutó la aplicación desde el IDE Netbeans 6.8, JDK 1.6.0_18, memoria heap configurada previamente con máximo de asignación de 256Mb, para las mediciones se obtienen los datos del Profile de Netbeans, el motor de base de datos es MySqlVersión 5.1.44. La base de datos seleccionada es la de pruebas con la que cuenta MySql y previamente poblada con 3,919,015 registros en las tablas.

2.3.2 Descripción de las Pruebas.Para el proceso de evaluación del rendimiento de los *framework*s seleccionados, se dispuso de la ejecución de las instrucciones básicas de SQL mostradas en la figura 3.

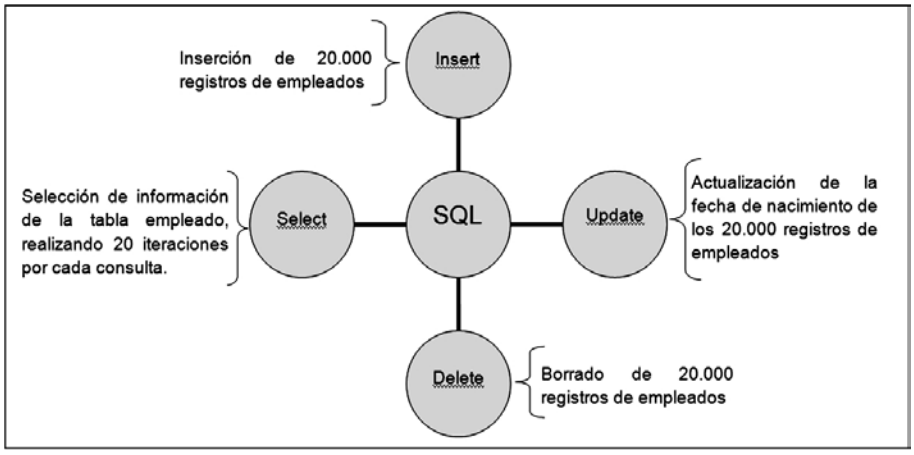

Figura 3. Descripción de las pruebas

Las consultas a la base de datos fueron:

- Seleccionar todos los empleados
- Seleccionar todos los empleados cuyo primer nombre sea “aamer”
- Seleccionar todos los empleados que hayan nacido después de 5 de febrero de 1946
- Seleccionar todos los empleados cuyo primer nombre empiece por “j”,
- Seleccionar todos los empleados que sea de género femenino
- Seleccionar todos los empleados que sean de género femenino que hayan ganado más de 60,000
- Obtener por consulta el número de empleados de cada departamento.

3. Resultados y discusión

Al finalizar las pruebas planteadas se obtuvo los siguientes resultados de acuerdo al cada comando aplicado.

3.1 Comparación durante inserciones

En la figura 4 se evidencia la diferencia en cuanto a consumo de memoria fue alta, *Hibernate* mantuvo un promedio de 7.4 Mb mientras que *EclipseLink* se fue elevando gradualmente hasta alcanzar 150 Mb, momento en el que dejó de funcionar saturando completamente la memoria *heap* configurada y si haber terminado el proceso (alcanzó a realizar 16.541 inserciones); por otra parte respecto al tiempo empleado

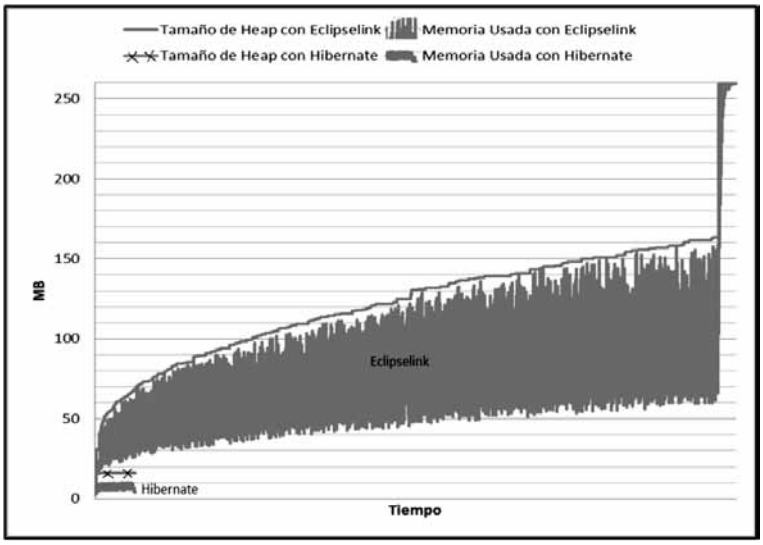

Figura 4. Comparación de memoria durante el proceso de inserción

en la ejecución de estas instrucciones *Hibernate* tardó 10 min 42.526 segundos frente a 3 horas 28 minutos 2.840 segundos de *EclpseLink* sin haber completado las peticiones configuradas.

EclipseLink mantuvo una constante de 14.4% de uso del tiempo para ejecutar el *Garbage Collector*, mientras que en *Hibernate* no alcanzó el 1%, lo cual permite afirmar que *Hibernate* tiene mejor administración de memoria en cuanto al proceso de inserción de registros.

3.2 Comparación durante Actualizaciones

De forma similar al comportamiento registrado en las inserciones, *Hibernate* realizó el proceso en menor tiempo (10 minutos 7.106 segundos) y con menor consumo de memoria (en promedio 7,2Mb), como se expone en la figura 5; mientras que nuevamente *EclipseLink* aumentó constantemente el uso de memoria alcanzando un total de 180 Mb finalizando en un tiempo total de 1 hora 0 minutos 10.001 segundos; es importante anotar que en este proceso *EclipseLink* logró completar las 20.000 actualizaciones previstas.

En cuanto al tiempo destinado para la ejecución del *Garbage Collector* en las actualizaciones, *EclipseLink* mantuvo un promedio de 4.5%, aunque tiene picos hasta del 10%; en este caso se puede concluir que para la ejecución de *update* el *framework* tiene una mejor administración de memoria, sin embargo aun sigue siendo alto comparado con *Hibernate* que nuevamente no alcanzó el 1%.

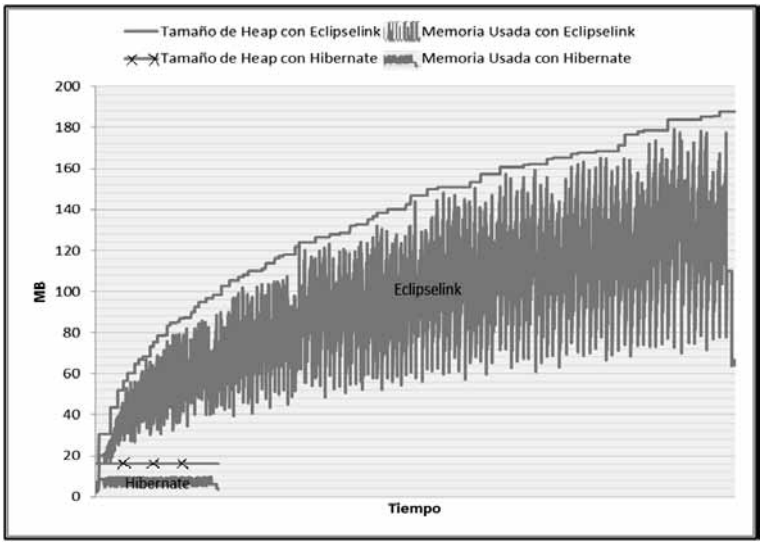

Figura 5. Comparación de memoria durante actualización de datos

3.3 Comparación durante Borrados

En el proceso para eliminar registros, respecto a tiempo de ejecución, se mantiene una diferencia aunque no es tan notoria como en las anteriores instrucciones, *Hibernate* desarrolló el proceso en 10 minutos 52.581 segundos, mientras que *EclipseLink* tardo 15 minutos 34.492 segundos. En lo referente a consumo de memoria sigue siendo innegable la efectiva administración de *Hibernate* que mantuvo un promedio de 7,5 Mb, mientras que *EclipseLink* mantiene la tendencia a aumentar constantemente hasta alcanzar en este caso 225 Mb, como se muestra en la figura 6.

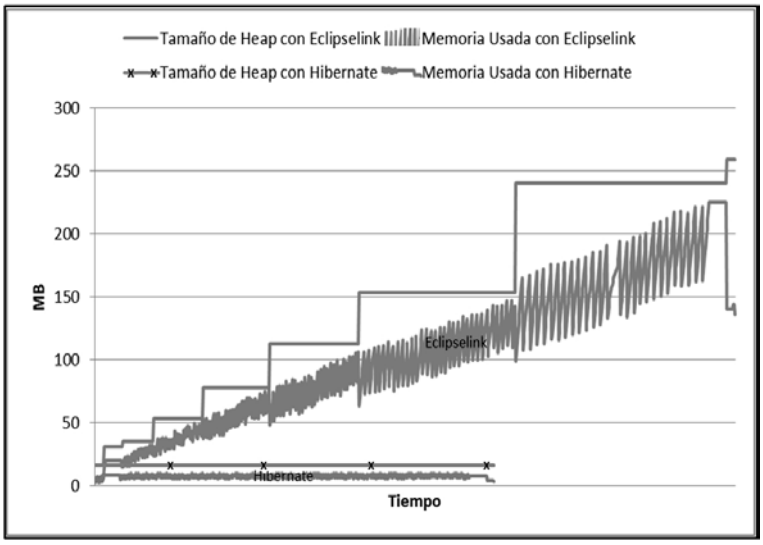

Figura 6. Comparación de memoria durante borrado de registros

Acorde a la notoria mejora de *EclipseLink* en el proceso de borrado de datos, el tiempo destinado para la ejecución del *Garbage Collector* mantiene un promedio de 1%, *Hibernate* continuó con un excelente resultado nuevamente sin alcanzar el 1%.

3.4. Comparación durante Consultas

En el proceso de consultas, utilizando lenguaje SQL para los dos *framework*, como se evidencia en las figuras 7 y 8, se mantuvo la superioridad de *Hibernate* aunque ya no es tan evidente, puesto que obtuvo un promedio de 124 Mb mientras que *EclipseLink* 126 Mb.

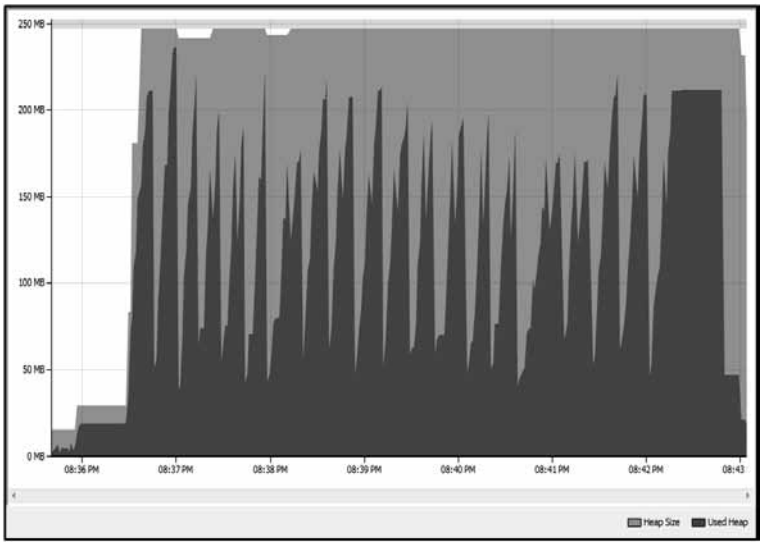

Figura 7. Memoria utilizada por EclipseLink en la ejecución de consultas

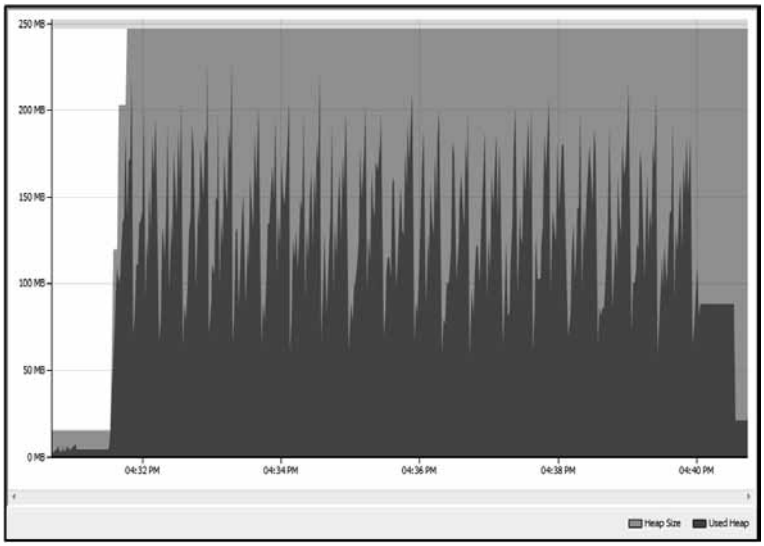

Figura 8. Memoria utilizada por Hibernate en la ejecución de consultas con SQL

El *framework* de persistencia *Hibernate*, cuenta con su propio Lenguaje nativo para consultas, denominado *HQL*, y contrario a lo esperado, al ejecutar las consultas utilizando *HQL*, estuvo más cerca de alcanzar el máximo de memoria *heap* asignado, con un promedio de 174 Mb, efecto que desmejoró notablemente el rendimiento.

El tiempo empleado para la recolección de datos basura es menor en *EclipseLink,* con un promedio de 14%, mientras que *Hibernate* por primera vez alcanzó un promedio de 16.1%.

El tiempo empleado para la recolección de datos basura es menor en *Hibernate,* utilizando SQL con un promedio de 16.1%, mientras que *Hibernate* con *HQL* alcanzó un promedio de 32.7%, este resultado permite deducir que contrario a lo esperado el uso de *HQL* en *Hibernate* no favorece su rendimiento.

4. Conclusiones

Aunque *EclipseLink* surge con base en *Toplink*, parece ser aún un *framework* poco estable y le falta mejorar en la administración de memoria y tiempo de respuesta.

Las pruebas permitieron observar que *Hibernate* es robusto, capaz de soportar cantidades inmensas de instrucciones SQL, sin afectar su rendimiento.

Partiendo del hecho de que los dos *framework* evaluados son proyectos de código libre, es evidente que este aspecto permite mantener un constante y rápido mejoramiento, puesto que su realimentación no se origina solo de los desarrolladores sino también de la comunidad que los utiliza.

A pesar que *Hibernate* se podría considerar un ganador dentro de la evaluación, las consultas realizadas con *HQL* tienen un desempeño bajo, aunque brinda facilidad al programador al momento de realizar consultas, le resta eficiencia al *framework* puesto que internamente debe realizar la traducción a lenguaje SQL.

Bibliografía

ACHARYA, Sharad. (2007). Adopting a Java Persistence *Framework*: Which, When, and What? [on line]. <http://today.java.net/pub/a/today/2007/12/18/adopting-java-persistence-*framework*.html>. [consult: 24/10/2010].

ALONSO AMO, Fernando; MARTÍNEZ NORMAND, Loïc y SEGOVIA PÉREZ, Francisco Javier. (2005). Introducción a la ingeniería del software. Madrid (España): Delta Publicaciones. 542 p. ISBN: 978-84-96477-00-1.

AYCART PÉREZ, David; GIBERT GINESTÁ, Marc; HERNÁNDEZ MATÍAS, Martín y HERNÁNDEZ, Jordi Mas (2007). Ingeniería del software en entornos de SL. 2 ed. Cataluña (España): Fundació per a la Universitat Oberta de Catalunya. 311 p. ISBN: 9788497077712.

BARNES, Jeffrey (2007). Object-Relational Mapping as a Persistence Mechanism for Object-Oriented Applications. Tesis de Maestría (Magister in Computer Science). Saint Paul (Minnesota, USA): Macalester College. 113 p.

BENJELLOUN, Rida (2006). Archimède: A Canadian solution for institutional repository. En: Library Hi Tech. Vol. 23. No. 4. Québec (Canada): Université Laval. p. 481-489. ISSN: 0737-8831

ECLIPSELINK. (2011). Definición [en línea]. Ottawa (Ontario, Canada): The Eclipse Foundation. <http://www.eclipse.org/eclipselink/>. [Consulta: 10/02/2011].

GOLDSCHMIDT, Thomas; REUSSNER, Ralf and WINZEN, Jochen (2008). A case study evaluation of maintainability and performance of persistency techniques. En: Proceedings of the 30th international conference on Software engineering (ICSE '08). New York (EUA): ACM. p. 401-410. ISBN: 978-1-60558-079-1

GOSLING, James; JOY, Bill; STEELE, Guy and BRACHA, Gilad (2005). Java (Tm) Language Specification. 3 ed. Boston (EUA): Addison-Wesley Professional. 645 p. ISBN 0-321-24678-0

HEMRAJANI, Anil (2006). Agile Java Development with Spring, Hibernate and Eclipse (Developer's Library). Indianápolis (USA): Sams. 334 p.ISBN: 9780672328961.

HIBERNATE (2009). Foros de la comunidad Hibernate [on line]. s.l.: Red Hat, Inc. <https://forum.hibernate.org/>. [consulta: 02/12/210].

HIBERNATE (2011). Relational Persistence for Java and .NET. [on line]. s.l.: Red Hat, Inc. <https://www.hibernate.org/>. [consulta: 02/01/2011].

JOHNSON, Ralph. and FOOTE, Brian (1988). Designing Reusable Classes. In: Journal of Object-Oriented Programming, Vol. 1, No. 2. Denville (USA): SIGS Publications. p. 22–35. ISSN: 0896-8438.

KALANTARI, Reza and BRYANT, Christopher (2009). Comparing the Performance of Object and Object Relational Database Systems on Objects of Varying Complexity. [on line]. Salford (Manchester, UK): School of Computing, Science and Engineering, Newton Building, University of Salford. <http://www.reza-kalantari.com/files/paper_camera.pdf>. [consult: 20/12/2010].

KING, Gavin; BAUER, Christian; ANDERSEN, Max Rydahl; BERNARD, Emmanuel y EBERSOLE, Steve (2011).Hibernate - Persistencia relacional para Java idiomático. [en línea]. s.l.: Red Hat, Inc. <http://docs.jboss.org/hibernate/core/3.5/reference/es-ES/pdf/hibernate_reference.pdf>. [consulta: 10/01/2011].

NOLTE, Jörg (1995). Towards a Persistence *Framework* for High Performance Computing Systems [on line]. Pennsylvania (USA): The Pennsylvania State University <http://citeseerx.ist.psu.edu/viewdoc/summary?> [consult: 10/11/2010].

O'NEIL, Elizabeth (2008). Object/relational mapping 2008: hibernate and the Entity Data Model (EDM) [on line]. In: International conference on Management of data, SIGMOD '08. (9-12/06/2008), Vancouver (Canada): ACM. SIGMOD '08 Proceedings of the 2008 ACM SIGMOD international conference on Management of data, New York (USA): ACM Digital Library, p. 1351-1356. ISBN: 978-1-60558-102-6. <http://portal.acm.org/ft_gateway.cfm?id=1376773&type=pdf> [consult: 16/12/2010]

ORACLE (2011). Oracle TopLink 11g [on line]. Redwood Shores (CA, USA): Oracle Corporation. <http://www.oracle.com/technetwork/middleware/toplink/overview/index.html> [consulta: 10/01/2011].

PRESSMAN, Roger. (2010). Ingeniería del Software; un enfoque práctico. 7 ed. México D.F. (México): McGraw-Hill. 900 p. ISBN: 978-607-15-0314-5.

RICHARDSON, Chris (2009). ORM in Dynamic Languages. In: Communications of the ACM - A Direct Path to Dependable. Vol. 52, No. 4 (April). New York (USA): ACM. p. 48-55. ISSN: 0001-0782

SUN MICROSYSTEMS (2008). What is garbage collection. [on line]. <http://www.java-tips.org/java-se-tips/java.lang/what-is-garbage-collection.html> [consult: 15/12/2010].

SUN MICROSYSTEMS (2010). Java Persistence API [on line]. <http://java.sun.com/javaee/technologies/persistence.jsp>.[consult: 5/01/2010].

THROUGHPUT.IBM (2010). Measurements of performance [on line]. Armonk (NY, USA): .IBM Corporation. <http://publib.boulder.ibm.com/infocenter/idshelp/v115/topic/com.ibm.perf.doc/ids_prf_041.htm>.[consult: 22/12/2010].

TUYA GONZALEZ Javier; RAMOS ROMAN Isabel y DOLADO COSÍN, Javier. (2007). Técnicas cuantitativas para la gestión en la ingeniería del software: Técnicas y prácticas en las pruebas de software. Madrid (España): Netbiblo. 373 p. ISBN: 9788497452045.

VASA Rajesh; LUMPE, Markus and SCHNEIDER, Jean-Guy (2007). Patterns of component evolution. In: International conference on Software composition (SC'07).Berlin (Germany): Springer-Verlag. p. 235-251. ISBN: 978-3-540-77350-4.

WEI Fuguo y LEE Sai (2009). A Mapping Mechanism for Objects Persistence in Multivalued Database. In: 2nd IEEE International Conference on Computer Science and Information Technology. Beijing (China): IEEE, p. 292-296. ISBN: 978-1-4244-4519-6.

WHITE Jim. (2010). Sizing Up Open Source Java Persistence. [on line]. Foster City (CA, USA): Internet.com. <http://www.devx.com/ Java/Article/33768/1954>. [consult: 10/11/2010].

ZYL, Pieter Van; KOURIE, Derrick G; COETZEE, Louis and BOAKE, Andrew (2009). The influence of optimizations on the performance of an object relational mapping tool. En: Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT '09). New York (USA): ACM. p. 150-159. ISBN: 978-1-60558-643-4.

ZYL, Pieter Van; KOURIE, Derrick; COETZEE, Louis and BOAKE, Andrew (2006). Comparing the performance of object databases and ORM tools. In: Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries. South African (South African): Institute for Computer Scientists and Information Technologists. p. 1-11. ISBN: 1-59593-567-3.