

Índice de contenido

1. Abstract.....	2
2. Motivación / Propósito.....	2
3. Estado del arte.....	3
4. Plan de la obra.....	6
4.1. Metodología.....	6
5. Desarrollo.....	7
5.1. Introducción.....	7
5.2. Breve descripción de la aplicación de prueba.....	9
5.3. Arquitectura general.....	12
5.3.1. Propiedades de arquitectura.....	12
5.3.2. Patrones de arquitectura y diseño.....	14
5.3.3. Arquitectura física propuesta.....	18
5.4. Descripción de las herramientas.....	20
5.4.1. JBoss AS.....	20
5.4.1.1. Herramientas similares.....	20
5.4.2. Terracotta + Tomcat.....	21
5.4.2.1. Herramientas similares.....	22
5.5. Pruebas.....	23
5.6. Problemas encontrados y soluciones propuestas.....	35
5.6.1. Generales.....	35
5.6.2. JBoss.....	49
5.6.3. Terracotta + Tomcat.....	54
5.7. Resultados.....	57
5.7.1. JBoss.....	59
5.7.2. Terracotta + Tomcat.....	75
6. Conclusiones.....	93
7. Apéndices.....	96
7.1. Apéndice 1: Cómo utilizar las herramientas seleccionadas.....	96
7.2. Apéndice 2: Breve descripción de los scripts creados.....	103
7.3. Apéndice 3: Detalle del manejo de transacciones.....	105
7.4. Apéndice 4: Diferencias en la implementación de la sincronización.....	106
7.5. Apéndice 5: Código de la clase PojoCacheManagerImpl, utilizada en el Cluster con JBoss.....	113
7.6. Apéndice 6: Ejecución de la combinación de casos de pruebas manuales.....	119
8. Datos de los autores.....	144
Barrabino, Diego.....	144
Moreyra Martín.....	149
9. Bibliografía.....	152

1. Abstract

El objetivo del presente trabajo, es analizar y comparar dos arquitecturas con soporte de Clustering, a partir del desarrollo y utilización de una aplicación Web de prueba, que brinde escalabilidad, alta disponibilidad y alto desempeño, junto con la estructuración, configuración, administración y despliegue del entorno productivo correspondiente. Particularmente, los dos tipos de arquitecturas a comparar utilizan las herramientas JBoss y Terracotta + Tomcat, respectivamente.

2. Motivación / Propósito

A medida que van apareciendo más aplicaciones web integrales que resuelven cada vez más y más aspectos de diferentes tipos de negocio, más se va dependiendo de dichas soluciones, y más impacto tienen las caídas de los sistemas. Por lo cual, tener solamente una aplicación web atractiva y útil de acuerdo a ciertos fines, algo para nada fácil de lograr por cierto, es solo una parte del desafío si se quiere desarrollar una aplicación realmente exitosa; otra parte del desafío es lograr que esté disponible prácticamente siempre.

Una de las soluciones posibles para intentar que las aplicaciones estén en línea la mayor cantidad de tiempo posible, y particularmente la solución que nos interesa, es implementar Clustering en el entorno productivo de la aplicación. De esta forma, por las características propias de la técnica, se puede mantener la aplicación en línea incluso en caso de que alguna de las computadoras que conforman el Cluster deje de funcionar, entre otros inconvenientes que podrían surgir.

Revisando las tecnologías disponibles, vemos que podríamos montar ambientes de tales características utilizando herramientas populares en el mercado, como ser el servidor JBoss, o una librería de Java denominada Terracotta en conjunción con el contenedor de servlets Tomcat. También notamos que las herramientas elegidas proponen dos maneras distintas de abordar la implementación de un ambiente de Clustering, lo cual aporta a la riqueza de la investigación.

3. Estado del arte

El término Cluster usualmente se aplica a los conjuntos o conglomerados de computadoras contruidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora.

Sin embargo este término se ha llevado al contexto de las aplicaciones empresariales de tipo cliente - servidor, en donde un Cluster es un conjunto de instancias del servidor que desde el punto de vista de los clientes, se comportan como un servidor único. Cada instancia del servidor se puede encontrar tanto en la misma computadora, como así también en múltiples computadoras conectadas entre si por medio de una red; estos servidores físicos no necesariamente deben compartir el mismo tipo de hardware.

La tecnología de Clusters ha evolucionado en apoyo de actividades que van desde aplicaciones de supercómputo y software de misiones críticas, servidores web y comercio electrónico, hasta bases de datos de alto rendimiento, entre otros usos.

De un Cluster se espera que presente las siguientes propiedades de arquitectura [FIELDING]:

- 1.- Alto rendimiento.
- 2.- Alta disponibilidad.
- 3.- Escalabilidad.

La construcción de los servidores del Cluster es fácil y económica debido a su flexibilidad: pueden tener todos la misma configuración de hardware y sistema operativo (Cluster homogéneo), diferente rendimiento pero con arquitecturas y sistemas operativos similares (Cluster semi-homogéneo), o tener diferente hardware y sistema operativo (Cluster heterogéneo), lo que hace más fácil y económica su construcción. Incluso, pueden montarse todos los nodos de un Cluster en una misma computadora (topología de Cluster vertical; práctico cuando se tiene a disposición una supercomputadora), o puede montarse cada nodo en una computadora distinta (topología de Cluster horizontal; práctico cuando las computadoras a disposición no dejan de ser computadoras o servidores normales), o pueden montarse los nodos según combinaciones de estas dos topologías, para un óptimo uso del hardware a disposición [JBOSS AS IN ACTION].

Sin embargo, para que un Cluster funcione como tal, no basta sólo con conectar entre sí los servidores, sino que es necesario proveer un sistema de manejo del Cluster, el cual se encargue de interactuar con el usuario y los procesos que corren en él para optimizar el funcionamiento. A continuación se presentan dos herramientas que cuentan con gran popularidad y que pueden realizar estas funciones:

- JBoss: propone replicar partes de una aplicación en varios servidores sincronizándolos mediante mecanismos provistos por él mismo. Una aparente desventaja de esta herramienta es que requiere efectuar implementaciones específicas requeridas por el servidor de aplicaciones, condicionando el diseño de la aplicación. Sin embargo esto tiene una gran ventaja, ya que se al desarrollar una aplicación específicamente pensada para funcionar con esta herramienta, probablemente se obtenga una muy buena performance [CLUSTERED JAVA EE].
- Terracotta (junto con Tomcat): provee de un conjunto de herramientas, con el fin de montar una aplicación pensada para un único servidor y desplegarla en múltiples servidores de manera que no tengan conocimiento de existencia entre si. El encargado de la sincronización es justamente el conjunto de librerías de Terracotta. Una aparente ventaja es que Terracotta no requiere implementaciones específicas para funcionar, por lo que en teoría se puede replicar cualquier aplicación. Por otro lado en contraposición con JBoss, esta situación lleva a una posible disminución de la performance [TERRACOTTA WORKS].

En otras palabras:

- JBoss provee una plataforma que fomenta pensar ciertas partes de la aplicación para que sean servicios, o para que estén clusterizadas, o para que estén distribuidas, o para que estén centralizadas o a nivel nodo o a nivel Cluster, etc. Quien diseñe una aplicación para JBoss, podrá incorporar o no esas facilidades a dicha aplicación; particularmente en nuestro caso, si bien hemos logrado que el impacto en el código sea mínimo, como se verá en las siguientes secciones, incorporamos las facilidades que posibilitan que ciertos componentes sean únicos en el contexto del Cluster. Lo que sería el armado en sí del Cluster, es posibilitado mediante los comandos con que se levantan los nodos que formarán el Cluster, no depende de la estructura de la aplicación.

- En cambio, Tomcat no provee una plataforma semejante a la de JBoss, si bien ha avanzado mucho desde sus primeras versiones, incluso al nivel de poder soportar por sí sólo conexiones con SSL y de proveer una forma de armar una aplicación distribuida en diferentes nodos, las funcionalidades provistas son muy limitadas, y no son comparables con las ofrecidas por JBoss. Por ende, el diseño de las aplicaciones que se utilizan con Tomcat es, se podría decir, un diseño clásico. Terracotta, en el uso que le estamos dando en este contexto de investigación, provee funcionalidades que administran los distintos deployments standalone que se hacen con Tomcat, y los hace funcionar como una aplicación clusterizada. Nuevamente, como en el caso de JBoss, hay un impacto en el código fuente de la aplicación, pero nuevamente como se aclarará en las siguientes secciones, es un impacto mínimo.

4. Plan de la obra

4.1. Metodología

La metodología elegida para llevar adelante este proyecto, consistió en definir los objetivos del mismo, y a partir de dichos objetivos, organizar el tiempo en secciones evolutivas del proyecto, a saber:

- 1.- Concepción.
- 2.- Elaboración.
- 3.- Análisis de la arquitectura.
- 4.- Construcción.
- 5.- Finalización.

Por la naturaleza del proyecto, las etapas más importantes son las primeras tres etapas.

A lo largo de todas las etapas, se preparó este informe integrador, que presenta información sobre todos los aspectos del trabajo.

5. Desarrollo

5.1. Introducción

El objetivo del trabajo es comparar una aplicación clusterizada con dos herramientas diferentes: JBoss y Terracotta + Tomcat. Esta comparación se realiza teniendo en cuenta las propiedades de arquitectura, desde diferentes puntos de vista:

- Las herramientas utilizadas para clusterizar la aplicación.
- Las métricas obtenidas de ambos Clusters.

El primer paso para poder realizar las comparaciones enunciadas, fue la búsqueda de una aplicación que necesitara de las propiedades de arquitectura vistas en [FIELDING]: buen rendimiento, escalabilidad y alta disponibilidad. Al decidir las características de la aplicación, elegimos un pequeño conjunto de casos de uso representativos, los cuales fueron implementados para ser clusterizados con ambas herramientas (ver sección "5.2. Breve descripción de la aplicación de prueba"). Sintetizando, la aplicación presenta funcionalidades que pueden ser ejecutadas en los distintos nodos del Cluster sin que medie sincronización entre ellas, y también presenta una funcionalidad que debe ser sincronizada de manera única en todo el Cluster al igual que las sesiones de usuario que también deben poder replicarse en todo el Cluster; en éste último caso naturalmente para que la aplicación pueda continuar dando servicio a los usuarios incluso luego de que colapse cualquier nodo del Cluster (lo cual será una de las pruebas a realizar). Esta decisión se tomó para poder poner a prueba los servicios de sincronización de cada tipo de Cluster.

Habiendo implementado la aplicación de prueba, la ajustamos para que no requiriese a nivel código fuente, modificaciones sustanciales que pudieran llevar a concluir que en realidad se compararon aplicaciones en esencia distintas. Por ejemplo, se evitó utilizar EJB independientemente de trabajar con un servidor JEE que las soporta (JBoss), e independientemente también de que en una etapa inicial del trabajo fueron contemplados. De haber utilizado EJB, nos habríamos encontrado con el dilema de incorporar o no algunas otras librerías a Terracotta + Tomcat (por ejemplo, OpenEJB -<http://openejb.apache.org/>-) para poder utilizar la misma aplicación en éste último Cluster, con el riesgo de corromper la naturaleza de la investigación al incorporarle al segundo Cluster librerías (las de, siguiendo con el ejemplo, OpenEJB) que compiten y se solapan en muchos aspectos con Terracotta. Es decir, el criterio detrás de la decisión de ajustar el código fuente de la aplicación, es posibilitar una comparación coherente de los resultados conseguidos sobre en esencia la misma aplicación, y permitir concluir con claridad que de encontrarse diferencias en los resultados, éstos se deben a las diferencias en la constitución del Cluster a partir de las herramientas elegidas.

Este ajuste fue aplicado en profundidad, salvo en el punto en que el servicio de sincronización de cada tipo de Cluster obligó a tener diferencias menores en el código, que no impactan en la arquitectura ni en el funcionamiento de la aplicación. Ver “7.4. *Apéndice 4: Diferencias en la implementación de la sincronización*”.

A continuación, definimos las pruebas automáticas a ejecutar sobre cada ambiente: pruebas de stress, carga, sólo sincronización, y de operación de todas las funcionalidades de la aplicación de pruebas salvo por las de sincronización. También definimos pruebas manuales, como por ejemplo desactivar cualquier nodo de cualquier Cluster.

Posteriormente, estructuramos cada uno de los ambientes con cada uno de los tipos de Clusters. Ver “7.1. *Apéndice 1: Cómo utilizar las herramientas seleccionadas*”.

El ambiente que armamos primero fue el de JBoss. Por practicidad tomamos la decisión de organizar todo el armado del ambiente de JBoss en una serie de scripts de bash; es mucho más fácil volver a ejecutar un script que acordarse de todos los aspectos de configuración que tienen que ver con el ambiente. Ver “7.2. *Apéndice 2: Breve descripción de los scripts creados*”.

Finalmente, ejecutamos la aplicación de pruebas en el ambiente de JBoss junto con el software para recolectar estadísticas (ver “7.1. *Apéndice 1: Cómo utilizar las herramientas seleccionadas*”), esto lo hicimos la cantidad de veces necesaria hasta que se completaron las pruebas, tanto sobre el Cluster como sobre un nodo individual, para poder determinar las diferencias frente a un uso normal de un servidor individual.

Hicimos exactamente lo mismo para el ambiente armado con Terracotta. Ver “7.1. *Apéndice 1: Cómo utilizar las herramientas seleccionadas*”.

En ambos casos, también se ejecutaron las pruebas manuales, como ser dar de baja un nodo, etc (ver “5.5. *Pruebas*”).

Luego de ver en acción a cada uno de los ambientes, pudimos determinar con claridad el impacto de las diferencias menores a nivel código fuente, de las diferencias de comportamientos de los dos tipos de Clusters, de las diferencias de métricas obtenidas, y de las diferencias de funcionamiento de los Clusters por un lado y de los nodos individuales por el otro.

Con todos estos elementos a disposición, formulamos las conclusiones finales que serán presentadas en las siguientes secciones del presente documento.

5.2. Breve descripción de la aplicación de prueba

La Aplicación Web que se considera para efectuar el análisis propuesto en el presente trabajo, es una aplicación administrativa para recibir trabajos y evaluar el desempeño académico de los alumnos de un instituto educativo de gran tamaño.

El escenario que se considera es el siguiente:

1.- Se provee un usuario del sistema a cada uno de los alumnos y a cada uno de los profesores, quienes luego incorporan su contraseña privada de acceso.

2.- Todos los alumnos están obligados, periódicamente, a incorporar al sistema todos los trabajos prácticos que van realizando a medida que transcurre el ciclo lectivo, junto con otra información pertinente que permite clasificar los trabajos prácticos de manera correcta. Particularmente, los trabajos prácticos podrán ser grupales, e incluso se los podrá confeccionar en este caso utilizando el sistema, en un esquema colaborativo. Ver "7.4. Apéndice 4: Diferencias en la implementación de la sincronización".

3.- En el instituto hay una gran cantidad de alumnos, y se espera que en la mayor parte de las materias se les solicite muchos trabajos prácticos, los cuales pueden llegar a ser considerablemente grandes. El uso diario del sistema es muy exigente, ya que todos los días existe un muy alto tráfico de trabajos, los cuales pueden ser creados en forma conjunta por los alumnos en forma colaborativa, creando y editando los trabajos prácticos al mismo tiempo, desde diferentes computadoras. Ver "7.4. Apéndice 4: Diferencias en la implementación de la sincronización".

4.- Los profesores deben estar constantemente evaluando el material cargado, ya que necesitan dicho material para determinar la condición de regularidad de los alumnos, para calificar los trabajos prácticos, y para que a fin de año se puedan obtener estadísticas generales del desempeño del alumnado. Los profesores deben terminar sus evaluaciones, ya sea de los trabajos o la general a fin de año, en determinadas fechas. Por lo cual cabe señalar que todos los días, algunos profesores deben obligatoriamente tener preparadas ciertas evaluaciones.

5.- El sistema debe estar siempre disponible, ya que si así no lo hiciera, los alumnos se verían impedidos de entregar a tiempo sus trabajos, lo que pondría en peligro su regularidad, al mismo tiempo que gran cantidad de profesores se verían impedidos de terminar sus evaluaciones a tiempo. El sistema debe ser escalable porque la dirigencia del instituto planea seguir expandiendo sus actividades, lo cual indefectiblemente hará crecer la aplicación y creará más presión sobre los servidores. El sistema debe mostrar

un altísimo desempeño, para poder sobrellevar que muchos alumnos incorporen sus trabajos al mismo tiempo, muchos profesores los consulten al mismo tiempo, que mientras que los alumnos cargan los profesores evalúen, etc.

Se propone para el presente trabajo profesional, considerar la implementación de una primera versión mínima del sistema explicado, que permita a los alumnos cargar un mínimo de información incluso colaborativamente, que permita a los profesores hacer una evaluación mínima de dicha información, y que permita en el futuro incorporar nuevos módulos como ser el módulo de generación de prioridades para inscripción, nuevas estadísticas generales, etc.. Esta implementación debe necesariamente cumplir con los requisitos no funcionales de escalabilidad, desempeño, y disponibilidad.

Elegimos un conjunto de casos de uso que pondrán a prueba las características deseadas de alta disponibilidad y buen desempeño, el cual se logra a través del Clustering.

Asumimos que el sistema debe ser utilizado por una universidad con gran cantidad de alumnos, por lo que la carga sobre el sistema resultante de cualquiera de los casos de uso mencionados a continuación será muy grande.

1.- Loguearse en el sistema: proveer acceso al sistema mediante usuarios que podrán tener acceso a las pantallas de carga o evaluación según corresponda.

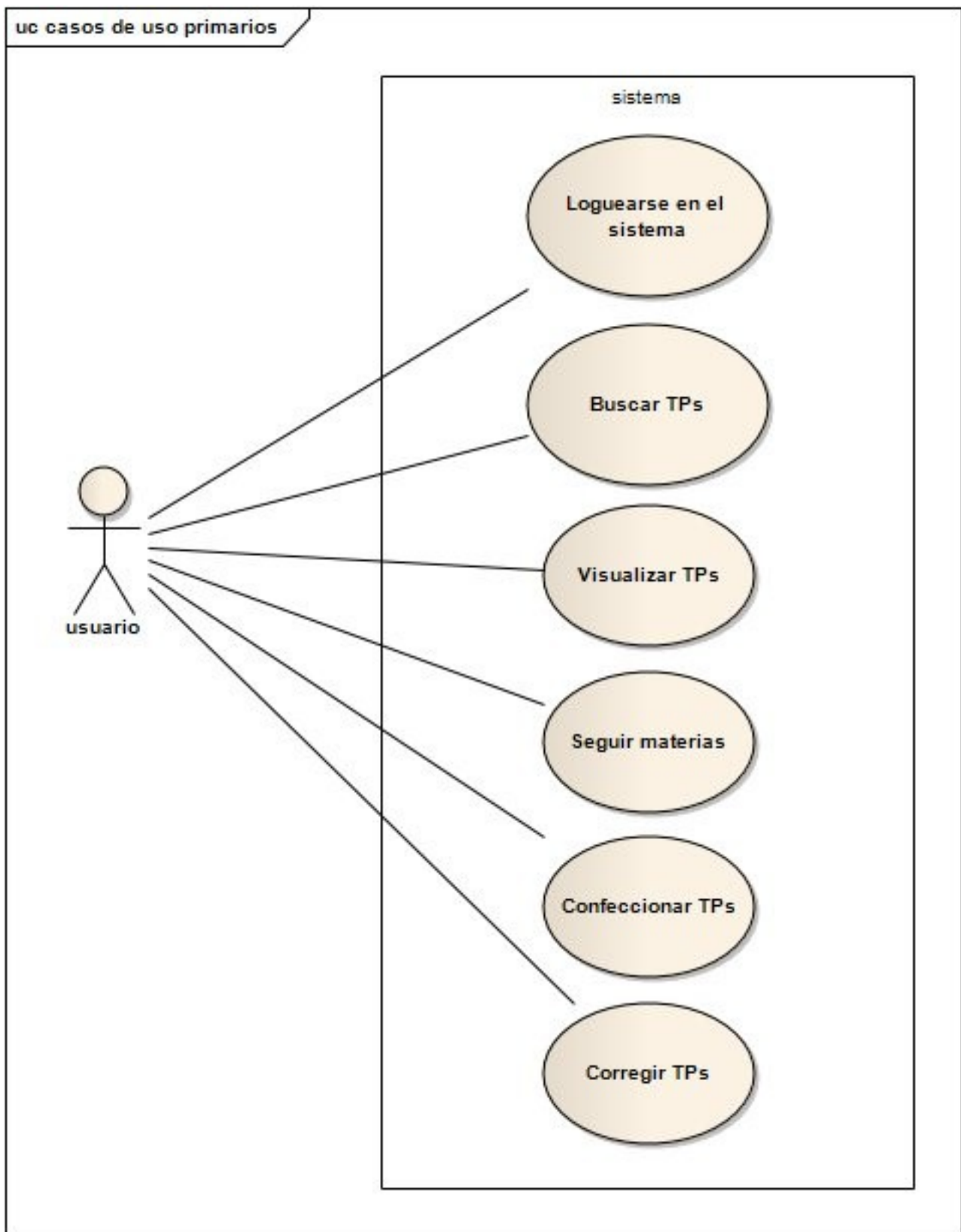
2.- Buscar TPs: pantalla de búsqueda de trabajos según cuatrimestre y materia.

3.- Visualizar TPs: pantalla de visualización de los trabajos cargados.

4.- Seguir materias (tiempo real): pantalla donde se mostrará dada una materia, la cantidad de TPs cargados, y se graficará la cantidad de TPs según su calificación, actualizando el gráfico cada 5 segundos.

5.- Confeccionar TPs (de manera colaborativa): varios alumnos de un trabajo práctico grupal pueden trabajar sobre el mismo trabajo práctico de manera colaborativa. Ver "7.4. Apéndice 4: Diferencias en la implementación de la sincronización".

Cabe aclarar, que el foco del presente trabajo no es esta aplicación en sí; el presente trabajo, no es un trabajo de programación de una aplicación para una institución educativa de gran tamaño, sino que el presente trabajo se centra en investigar y comparar arquitecturas de Cluster.

*Ilustración 1: Casos de uso primarios*

5.3. Arquitectura general

5.3.1. Propiedades de arquitectura

Durante el presente trabajo, nos referimos a la "arquitectura" de un sistema continuamente, por lo que es pertinente definir dicho concepto.

Empezaremos exponiendo la definición de arquitectura de software propuesta en [FIELDING]:

"La arquitectura de software queda definida por una configuración de elementos arquitectónicos -componentes, conectores y datos-, restringidos en sus relaciones con el propósito de obtener un conjunto de propiedades de arquitectura deseables".

Un componente es una unidad de software que está compuesta por instrucciones, estado e interfaces mediante las cuales se provee la funcionalidad de realizar transformaciones a datos. Estos componentes están relacionados por los conectores: mecanismos que median la comunicación, coordinación y cooperación entre los componentes.

Según Fielding la elección de una arquitectura significa elegir las restricciones que se aplicarán al sistema, y mediante la aplicación de estas restricciones se obtendrán las propiedades de arquitectura deseadas.

En concreto con respecto a nuestro sistema, la primera decisión sobre la arquitectura consistió en organizarla en capas en modo cliente - servidor. Esto introdujo un conjunto de restricciones, por ejemplo la forma de comunicación entre componentes, que finalmente nos proporcionó una mayor escalabilidad gracias a la separación de responsabilidades inducidas por la separación física de los componentes.

Recordemos las propiedades deseadas:

1.- Alto rendimiento (performance): esta propiedad se refiere a la velocidad de procesamiento del sistema. Esta velocidad se puede ver desde tres puntos de vista principales: el tiempo de respuesta, la latencia y el rendimiento.

Tiempo de respuesta: es el tiempo medio desde que se realiza un pedido al sistema, hasta que el mismo completa la tarea a realizar y envía una respuesta.

Latencia: el concepto de latencia refiere a la velocidad percibida por el usuario, desde que realiza un pedido, hasta que ve algún tipo de respuesta (sea la final o no).

Rendimiento: el concepto de rendimiento refiere a la cantidad de peticiones que puede manejar el sistema en un periodo de tiempo.

En nuestro caso nos concentraremos en los tiempos de respuesta medidos en milisegundos y el rendimiento, que mediremos en pedidos/segundo.

2.- **Alta disponibilidad:** esta propiedad refiere al grado de continuidad operacional del sistema. O sea que lo que se desea lograr es un sistema robusto, que siga operando inclusive bajo condiciones adversas para las cuales no fue pensado, como por ejemplo: picos en la cantidad de usuarios, congestión de la red, problemas con el hardware del servidor, etc.

3.- **Escalabilidad:** esta propiedad refiere a la habilidad de la arquitectura de soportar un gran número de componentes dentro de una configuración activa. Esto significa que a mayor carga, la arquitectura puede soportar un mayor número de componentes que manejen esta carga. Existen dos tipos de escalabilidad: vertical y horizontal. La escalabilidad vertical consiste en mejorar los recursos, por ejemplo mejorando el hardware utilizado. La escalabilidad horizontal consiste en aumentar la cantidad de recursos, o sea el agregado de componentes que puedan realizar procesamiento en paralelo. Esta definición marca un paralelismo con la definición de topología de Cluster, vertical y horizontal, vista anteriormente.

En nuestro caso nos enfocamos en la segunda opción: la escalabilidad horizontal. El objetivo del presente trabajo es justamente la implementación de un Cluster, basado en la replicación de nodos del servidor para poder procesar pedidos en forma paralela.

4.- Y por último podemos agregar un concepto relativamente nuevo en el área de las propiedades de arquitectura: la Elasticidad [CLOUD BEST PRACTICES]. Mientras que la escalabilidad identifica la posibilidad de aumentar o mejorar los recursos para manejar diferentes niveles de carga, la elasticidad se refiere a la facilidad y velocidad con la que se puede escalar la aplicación ante una variación de carga repentina (variación que puede ser predecible o impredecible).

5.3.2. Patrones de arquitectura y diseño

A nivel arquitectura elegimos una combinación de estilos arquitectónicos que nos proveerán de las propiedades enunciadas en la sección anterior, los estilos son: cliente - servidor en capas + repositorio replicado [FIELDING].

En primer lugar, una arquitectura en capas cliente - servidor, implica que va a existir un servidor que puede ejecutar cierta funcionalidad, y una cantidad variable de clientes, que pueden evolucionar en forma independiente del servidor, y que se comunicarán con éste realizando peticiones para ejecutar las diferentes funcionalidades provistas.

Por otro lado, apuntamos al estilo de repositorio replicado, que en esencia propone replicar servicios para poder proveerlos en mas de un servidor de forma paralela.

Para poder obtener una arquitectura alineada con los estilos enunciados, utilizamos una gran variedad de patrones, algunos de implementación directa y otros que son implementados por los frameworks utilizados. Estos patrones están basados en el libro Patrones Arquitectónicos de Aplicaciones Empresariales de Martin Fowler [PEAA].

Patrones arquitectónicos esenciales: este conjunto de patrones fueron los de implementación directa, son los patrones de lógica de dominio:

- Layers (Capas lógicas): la separación en capas nos permite desacoplar las distintas partes de la aplicación, pudiendo realizar cambios fácilmente en una capa sin afectar a otras. Otro beneficio de la separación en capas, es que dependiendo la herramienta utilizada, podemos separar la aplicación en diferentes paquetes, para ser distribuidos en diferentes servidores.
- Domain Model (utilizando Service Layer): refiere a modelar el problema con un enfoque puramente orientado a objetos que representan el modelo del problema a resolver. Y service layer es una capa que interactúa directamente con el modelo y provee la funcionalidad auxiliar necesaria para coordinar y acceder al modelo.

En el gráfico siguiente podemos observar la separación en capas, pero también se hace referencia a patrones de diseño específicos utilizados en cada capa.

- Capa de presentación: MVC (Model-View-Controller).
- Capa de lógica de negocio.
- Capa de persistencia: EAO (Entity-Access-Object).

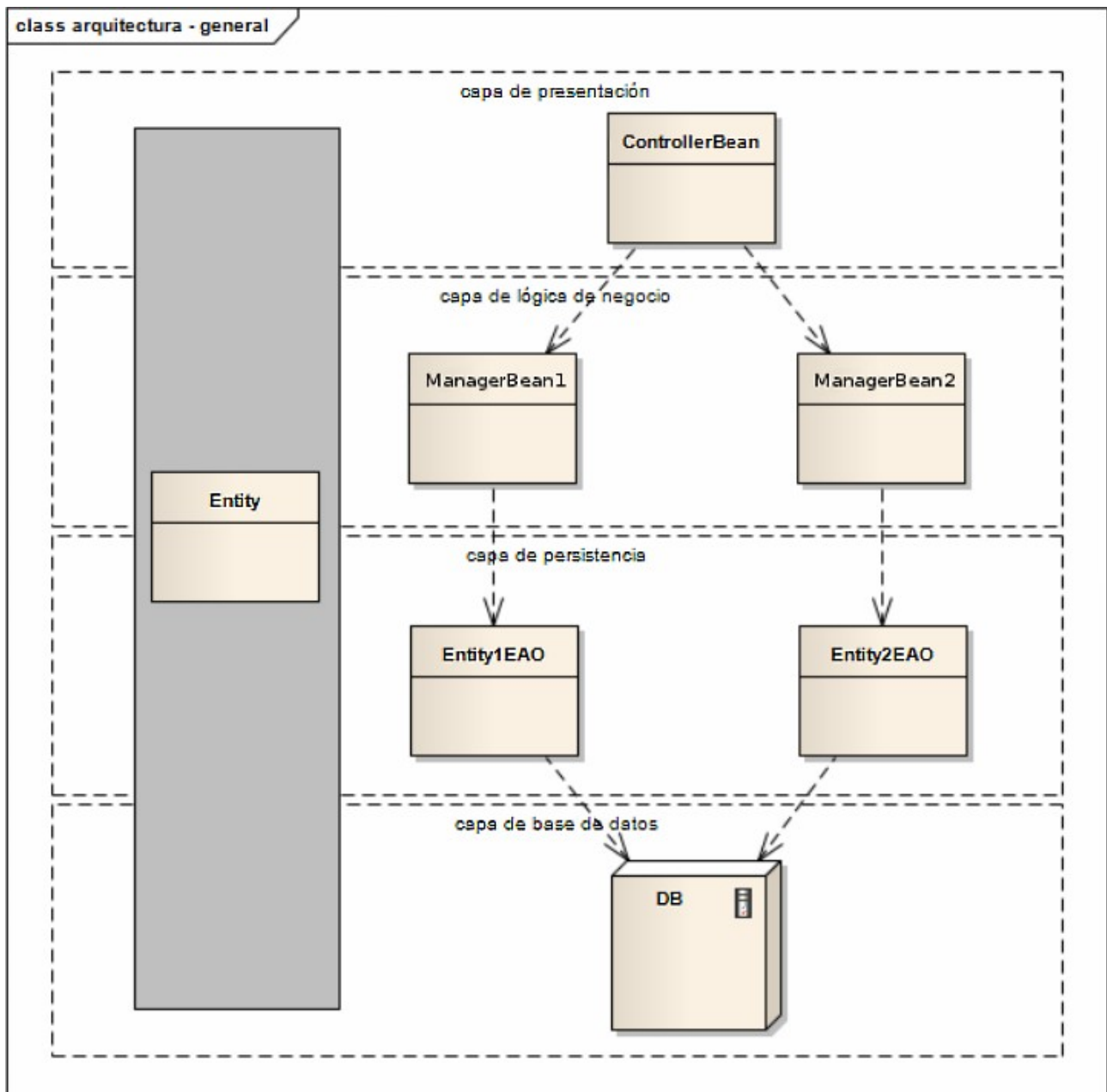


Ilustración 2: Arquitectura general

Presentación

La capa de presentación utiliza el patrón de diseño MVC, en particular utiliza el Framework SpringMVC (<http://www.springframework.org/>). Este framework es el encargado de manejar los pedidos (Request) de los navegadores, y hacer la llamada al Controlador correspondiente, decidiendo luego la vista a renderizar [SPRING MVC].

El proceso descrito se muestra a continuación:

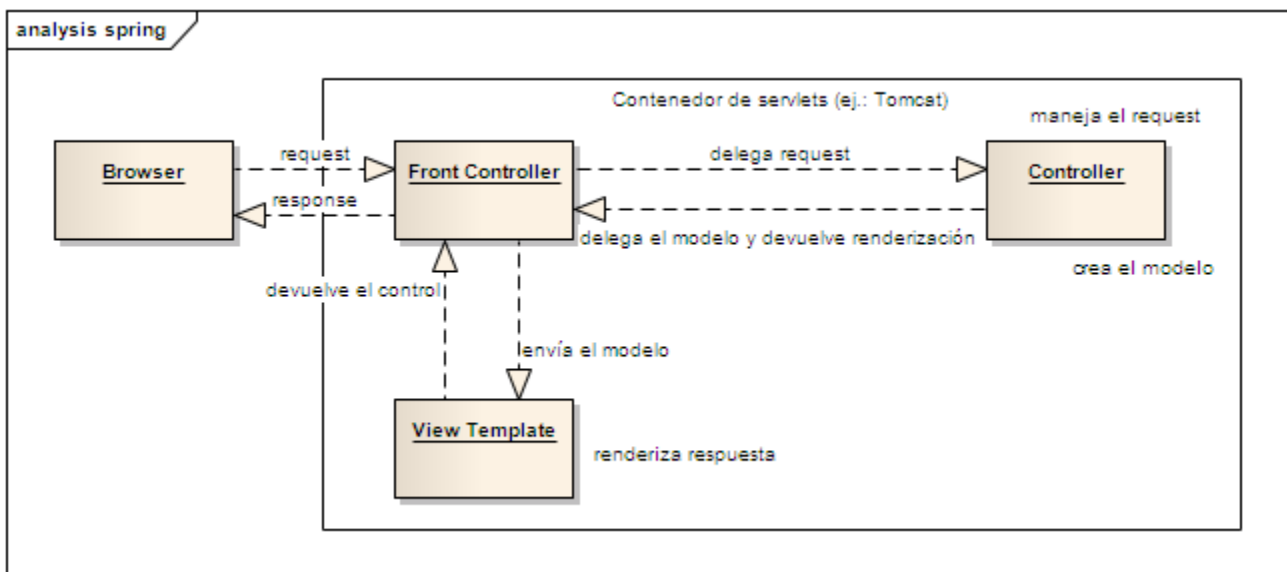


Ilustración 3: Presentación

Persistencia

Desde el punto de vista arquitectónico, se utilizaron los Patrones de Fuentes de Datos, de mapeo Objeto-Relacional de comportamiento y estructurales del [PEAA]: sin embargo este conjunto de patrones esta implementado directamente por el framework de persistencia utilizado: Hibernate. Dicho conjunto de patrones, está compuesto por:

- DataMapper.
- Lazy load.
- Unit of Work.
- Identity Map .
- Identity Field.
- Foreign Key Mapping.
- Association Table.

Desde el punto de vista de diseño, en la capa de persistencia se utiliza el patrón Entity-Access-Object (EAO) [EJB3 IN ACTION], este patrón provee una interfaz con las operaciones de persistencia, suficientemente versátil como para permitir trabajar correctamente en ambos servidores considerados.

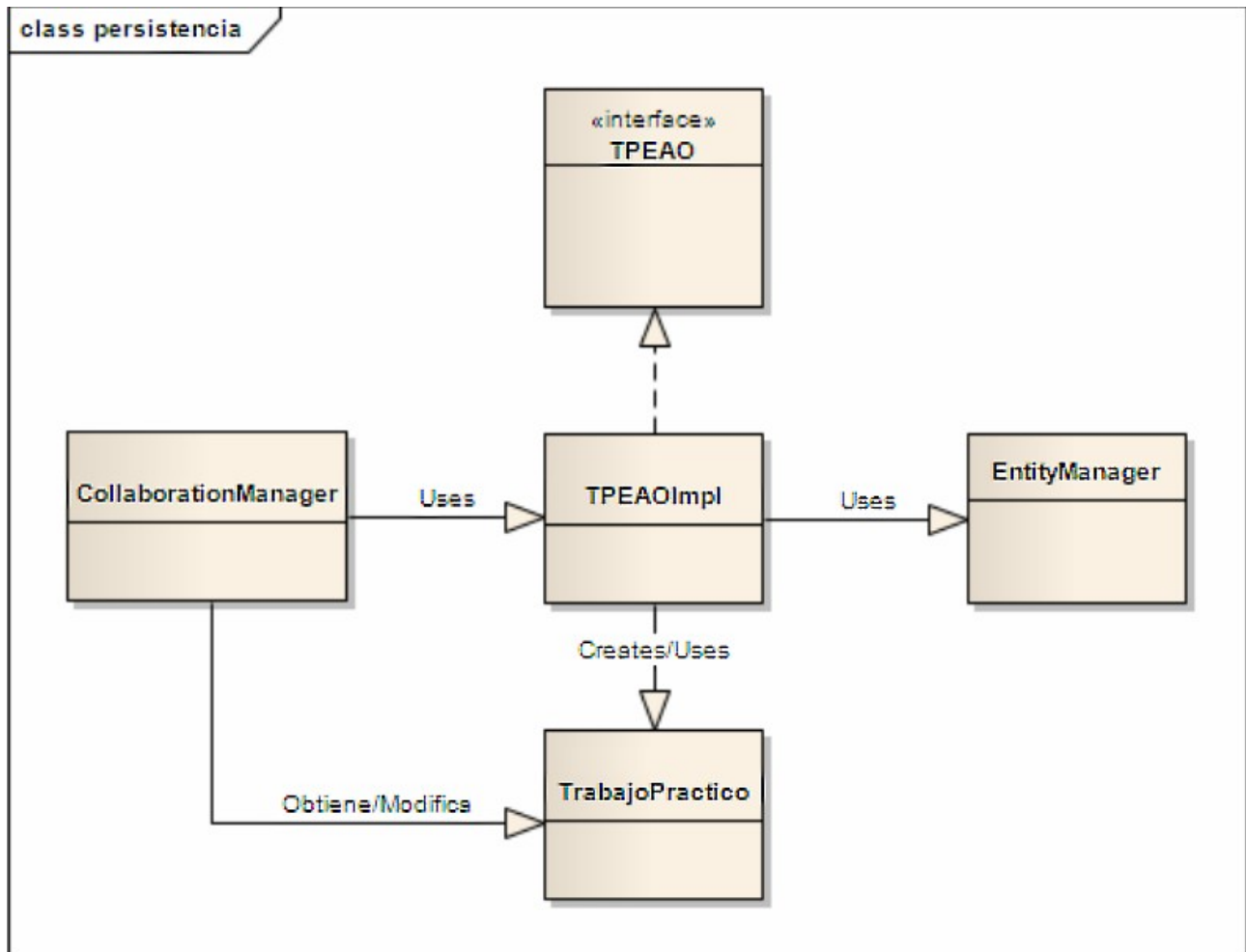


Ilustración 4: Persistencia

Como se puede ver en los gráficos anteriores, obviamos totalmente la utilización de clases auxiliares como Data Transfer Objects (DTO), ya que si bien las mismas agregarían un grado de abstracción, ante un cambio en el modelo se termina necesitando realizar el mismo cambio en los DTO, agregando complejidad innecesaria al desarrollo, sobre todo teniendo en cuenta que el objetivo del trabajo práctico no está relacionado con técnicas de desarrollo de aplicaciones.

5.3.3. Arquitectura física propuesta

Desde un punto de vista físico, proponemos la siguiente configuración de Cluster:

- Instancias del servidor: dos o más nodos.

En un contexto ideal se pueden poner en marcha dos nodos, con el fin de tener redundancia en caso de que uno de ellos falle, e ir activando nodos extra a medida que la carga de los mismos vaya aumentando. Por este motivo nos interesa la propiedad de elasticidad mencionada en el punto 5.3.1 de este documento.

- Balanceador de carga: un nodo.

El Cluster necesita de un punto de entrada inicial que recibirá los pedidos, y los despachará a los nodos del servidor correspondientes.

- Base de datos: un nodo.

No contemplamos una base de datos distribuida para no incrementar la complejidad del presente trabajo. Suponemos que el servidor con la base de datos no experimentará falencias ya que en un esquema productivo real, se pueden invertir parte de los recursos monetarios en infraestructura y hardware de mayor confianza para este servidor.

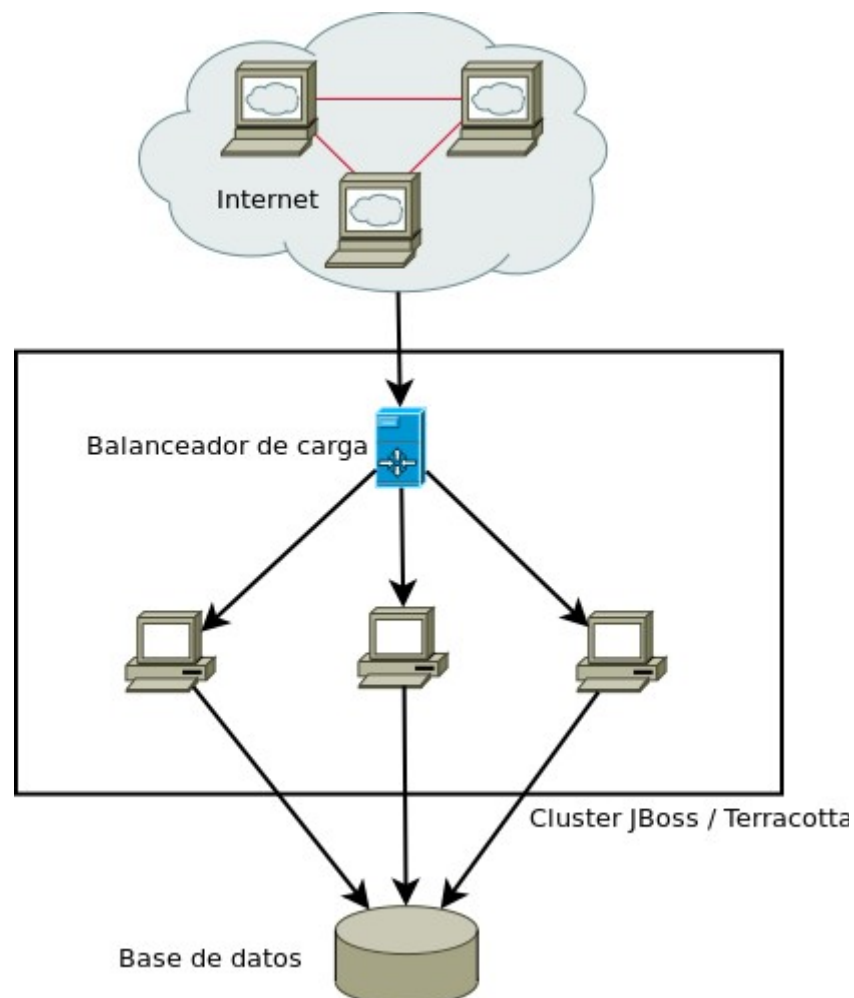


Ilustración 5: Estructura física propuesta

A partir de esta estructura física propuesta, modelamos nuestros Clusters de prueba, con los recursos disponibles.

5.4. Descripción de las herramientas

5.4.1. JBoss AS

URL: <http://www.jboss.org/jbossas/> ; <http://es.wikipedia.org/wiki/JBoss>

Versión: 6.1.0.Final

Descarga: <http://download.jboss.org/jbossas/6.1/JBoss-as-distribution-6.1.0.Final.zip>

JBoss es un servidor de aplicaciones J2EE de código abierto implementado puramente en Java; por ende, JBoss puede ser utilizado en cualquier sistema operativo para el que esté disponible Java [JBoss AS INSTALLATION]. Los principales desarrolladores trabajan para una empresa de servicios, JBoss Inc., adquirida por Red Hat en abril del 2006, fundada por Marc Fleury, el creador de la primera versión de JBoss. El proyecto está apoyado por una red mundial de colaboradores.

Este servidor implementa todo el paquete de servicios de J2EE.

JBoss soporta Clusterización de manera nativa. Simplemente, al levantar los servidores que conforman el Cluster, se ejecuta un comando que los sincroniza y logra que se comporten en conjunto como si fueran un solo servidor [CLUSTERED JAVA EE - START].

Por ejemplo, imaginemos un Cluster de tres nodos, cada uno de ellos en una computadora distinta.

Ver "7.1. Apéndice 1: Cómo utilizar las herramientas seleccionadas".

5.4.1.1. Herramientas similares

- Glassfish: <http://glassfish.java.net/es/>
- Apache Geronimo: <http://geronimo.apache.org/>
- Cualquier servidor de aplicaciones que implemente Java EE:
http://es.wikipedia.org/wiki/Java_EE#Servidores_de_Aplicaciones_Java_EE_5_certificados

5.4.2. Terracotta + Tomcat

Terracotta

URL: <http://terracotta.org/>

Versión: 3.6.0

Descarga: <http://terracotta.org/downloads/enterprise-ehcache?destination&name=terracotta-ee-3.7.0-installer.jar&bucket=tcdistributions&file=terracotta-ee-3.7.0-installer.jar>

Terracotta es un software de código abierto que permite crear aplicaciones Java que pueden escalar a cualquier cantidad de computadoras, sin tener que crear código adicional o usar bases de datos para compartir datos dentro del Cluster.

Terracotta utiliza el concepto de Memoria Adjunta a la Red (NAM - Network Attached Memory). El uso de NAM le permite a Terracotta distribuir en Cluster a Máquinas Virtuales Java (JVM) directamente debajo de las aplicaciones, brindando alta disponibilidad y escalabilidad de forma transparente.

La ventaja de Terracotta es que la aplicación preparada para funcionar en Cluster es exactamente igual a una aplicación Java común. Todos los conceptos y librerías que se usan (POJOs, Spring, Hibernate, threads, sincronización, etc.) funcionan de la misma manera con Terracotta en un entorno distribuido, de la misma manera que funcionan en una única máquina virtual con muchos hilos de ejecución.

Terracotta funciona a nivel de memoria, por lo que no es necesario heredar de ninguna clase ni implementar ninguna interfaz para compatir objetos entre todas las máquinas virtuales del Cluster (ni siquiera es necesario implementar `java.io.Serializable`).

En definitiva, se puede programar la aplicación de manera natural, y se deja a Terracotta que administre todo el trabajo para crear un entorno escalable y de alta disponibilidad [TERRACOTTA WORKS].

Ver "7.1. Apéndice 1: Cómo utilizar las herramientas seleccionadas".

Tomcat

URL: <http://tomcat.apache.org/>

Versión: 7.0.23

Descarga: <http://apache.dattatec.com/tomcat/tomcat-7/v7.0.30/bin/apache-tomcat-7.0.30.tar.gz>

Tomcat es un Servlet Container, el cual provee el entorno de ejecución de servlets, mediante los cuales se puede programar una aplicación web.

Los mencionados servlets son simples clases de Java, que reciben pedidos HTTP y envían respuestas que son mostradas en el navegador web del cliente.

Si bien Tomcat y Terracotta son herramientas totalmente diferentes, inclusive de diferentes compañías, éstas se complementan, ya que se puede desarrollar una aplicación web al estilo tradicional utilizando Tomcat y sin tener en cuenta aspectos de la Clusterización, y luego utilizar Terracotta para replicar esta aplicación web y lograr un comportamiento de Cluster.

Ver “7.1. Apéndice 1: Cómo utilizar las herramientas seleccionadas”.

5.4.2.1. Herramientas similares

Terracotta

- Oracle Coherence: <http://www.oracle.com/technetwork/middleware/coherence/overview/index.html>
- GridGain: <http://www.gridgain.com/>

Tomcat

- Jetty: <http://jetty.codehaus.org/jetty/>

5.5. Pruebas

Herramienta dedicada: Apache JMeter

URL: <http://jmeter.apache.org/>

Versión: 2.7

Descarga: <http://apache.xfree.com.ar//jmeter/binaries/apache-jmeter-2.7.tgz>

La herramienta Apache JMeter es una aplicación de escritorio hecha puramente en Java y totalmente open source, diseñada para ejecutar pruebas de carga y mediciones de desempeño sobre una aplicación, en nuestro caso una aplicación web, objetivo para el cual fue originalmente pensado JMeter, aunque ahora se ha diversificado.

Esta herramienta nos permite salvar un “plan de pruebas” creado directamente sobre la GUI web de la aplicación, con la posibilidad de ejecutarlo la cantidad de veces que se desee, y pudiendo variar diversos parámetros como ser: tiempo entre pedidos (requests), cantidad de usuarios concurrentes, parámetros necesarios para ejecutar la prueba como ser nombres de usuario, contraseñas, etc.

Dedicamos un plan de prueba a cada una de las pruebas automáticas que definimos, que se explicarán a continuación, y justamente ejecutándolos varias veces modificando el parámetro de cantidad de usuarios, es que hemos podido recolectar la información con la que hemos llegado hasta las conclusiones finales de presente trabajo.

Ver “7.1. Apéndice 1: Cómo utilizar las herramientas seleccionadas”.

General

Las pruebas buscan medir las diferentes propiedades de arquitectura expuestas. Para esto definimos dos tipos de pruebas: manuales y automáticas. Las pruebas manuales son pruebas generales sobre el Cluster, mientras que las pruebas automáticas son específicas para obtener una medición de los tiempos de respuesta y rendimiento según diferentes niveles de carga.

En primer lugar debemos explicitar los tiempos de respuesta estándar para una aplicación web:

Descripción de la situación	Limite tiempo
Usuarios manipulando objetos en forma directa en la UI	0.1 segundo
Usuarios navegando la aplicación	1 segundo
Mostrar gráficos y reportes	10 segundos
Aceptar y procesar toda la entrada del usuario	10 segundos

Estos tiempos son tiempos aproximados y no necesariamente significa que si la respuesta tarda mas del límite es inaceptable, sino que son tiempos de espera deseados, y bajo los cuales no se pierde la atención del usuario, ya que el mismo siente cierta fluidez en la interacción con la aplicación. Dichos tiempos pueden variar, pero deberían en promedio mantenerse por debajo de los límites citados.

La tabla de tiempos es estándar en cualquier aplicación, en particular en una aplicación web no se espera ninguna variación con respecto a tiempos en una aplicación local [RESPONSIVENESS].

Entorno de pruebas

Teniendo en cuenta lo anterior, definimos el entorno de pruebas, esto es debido a que no poseemos el hardware ni la infraestructura necesarias para montar un entorno productivo real. Sin embargo, cabe notar que el entorno de pruebas está inspirado en el entorno productivo real propuesto en este trabajo.

Disponemos de tres computadoras:

- Netbook: Acer Aspire One
(http://www.acer.com/aspireone/aspireone_8_9/), procesador Intel(R) Atom(TM) CPU N270 @ 1.60GHz, placa madre Acer, placa de red Realtek RTL8102E/RTL8103E Family PCI-E Fast Ethernet NIC, placa de red wifi Atheros AR5007EG Wireless Network Adapter, memoria RAM de 2 plaquetas de 512 Mb DDR2 (PC2-5300) 333 MHz (DDR2 667), disco rígido Hitachi HTS543216L9A300 160.0 GB 5400 RPM Serial ATA, sistemas operativos Windows XP y Ubuntu Linux 12.04 en dual boot.
- Notebook I5 4GB RAM: Positivo BGH A 490
(<http://www.positivobgh.com.ar/#/productos> => Xpert Ultra A-400 => Xpert Book A-490), procesador Intel(R) Core I5(TM)-2410M @ 2.30Ghz x 4, placa madre Intel, placa de red Realtek RTL8111/8168B PCI Express Gigabit Ethernet, placa de red wifi Realtek RTL8188CE 802.11b/g/n WiFi Adapter, memoria RAM de 4 Gb SODIMM DDR3 1334 Mhz, disco rígido Samsung HM641JI 640.0 GB Serial ATA, sistemas operativos Windows 7 y Ubuntu Linux 12.04 en dual boot.
- Notebook I3 4GB RAM: Positivo BGH A 470
(<http://www.positivobgh.com.ar/#/productos> => Xpert Ultra A-400 => Xpert Book A-470), procesador Intel(R) Core I3(TM)-2410M @ 2.30Ghz x 4, placa madre Intel, placa de red Realtek RTL8111/8168B PCI Express Gigabit Ethernet, placa de red wifi Realtek RTL8188CE 802.11b/g/n WiFi Adapter, memoria RAM de 4 Gb SODIMM DDR3 1334 Mhz, disco rígido Samsung HM641JI 640.0 GB Serial ATA, sistemas operativos Windows 7 y Ubuntu Linux 11.10 en dual boot.

Por cuestiones de practicidad, en etapa de desarrollo, en lugar de la Netbook se usó:

- Desktop: computadora clon, procesador Intel(R) Core(TM)2 Quad CPU Q8200 @ 2.33GHz, placa madre Intel DG41TY versión AAE47335-202, placa de red Realtek RTL8168D(P)/8111D(P) PCI-E Gigabit Ethernet NIC, memoria RAM de 2 plaquetas de 2 Gb DDR2-800 (400 MHz) SDRAM, disco rígido principal SAMSUNG HD501LJ 500 GBytes IDE SATA-II, disco rígido secundario SAMSUNG HD120IJ 120 GB IDE SATA-II, sistemas operativos Windows XP y Ubuntu Linux Desktop 12.04 en dual boot.

Por lo tanto armamos el entorno de pruebas de la siguiente forma:

- Instancias del servidor: dos nodos (Notebook I3; Notebook I5).
- Balanceador de carga: un nodo (Notebook I5).
- Base de datos: un nodo (Netbook / PC Desktop).
- Clientes: Utilizamos en cualquier computadora en que queramos interactuar con la aplicación, los navegadores de internet (Lynx -<http://lynx.browser.org/>-, Google Chrome / Chromium -<https://www.google.com/intl/es/chrome/browser/> ; <http://www.getchromium.org/>-, Mozilla Firefox -<http://www.mozilla.org/es-AR/firefox/new/>-) de la Notebook I3 y de la Notebook I5, y en el caso de las pruebas automáticas utilizamos una instancia de Apache JMeter 2.7 en cada máquina disponible.
- Instancias de JMeter: en todas las computadoras que forman parte del Cluster, bajo el criterio de que tener en una instancia de JMeter a 1200 usuarios, es equivalente a tener 3 instancias de JMeter con 400 cada una, siendo que es mucho más factible que en cualquier computadora haya recursos para una instancia de JMeter con 400 usuarios, que para una instancia de 1200 usuarios; por ende, como es mejor para acomodar, se utilizan instancias de JMeter en todas las computadoras, cada una con una fracción del total de usuarios que se necesita en la prueba.

Pruebas

Para armar los casos de prueba, confeccionamos la siguiente planilla, que instanciaremos en cada una de las pruebas definidas.

Descripción					
Clasificación		Aspecto			
		Disponibilidad	Carga	Performance	Distribución de cache
Capa de ejecución	Vista – Controlador (VC)				
	Controlador – Negocio (CN)				
Tipo de ejecución		[Manual / Automática]			
Entrada					
Salida					
Criterio de éxito					

Ilustración 6: Planilla de pruebas

Pruebas manuales

Las pruebas manuales encierran propiedades que son a nuestro criterio deseables en una estructura de Cluster de un ambiente productivo. Propiedades que de no cumplirse, harían que se perdiera el sentido de utilizar un Cluster.

Disponibilidad del sistema

Descripción		Disponibilidad incesante y automática	
Clasificación		Aspecto	
		Disponibilidad	
Capa de ejecución	CN	X	
Tipo de ejecución		Manual	
Entrada		Escenario con 2 nodos al que se conecta un cliente	
		Corte intencional de conexión de uno de los nodos	
Salida		Conclusión respecto a la disponibilidad	
Criterio de éxito		La aplicación siempre está disponible, aunque el cable que se retira sea del servidor preponderante en la configuración	

Ilustración 7: Planilla Disponibilidad del sistema

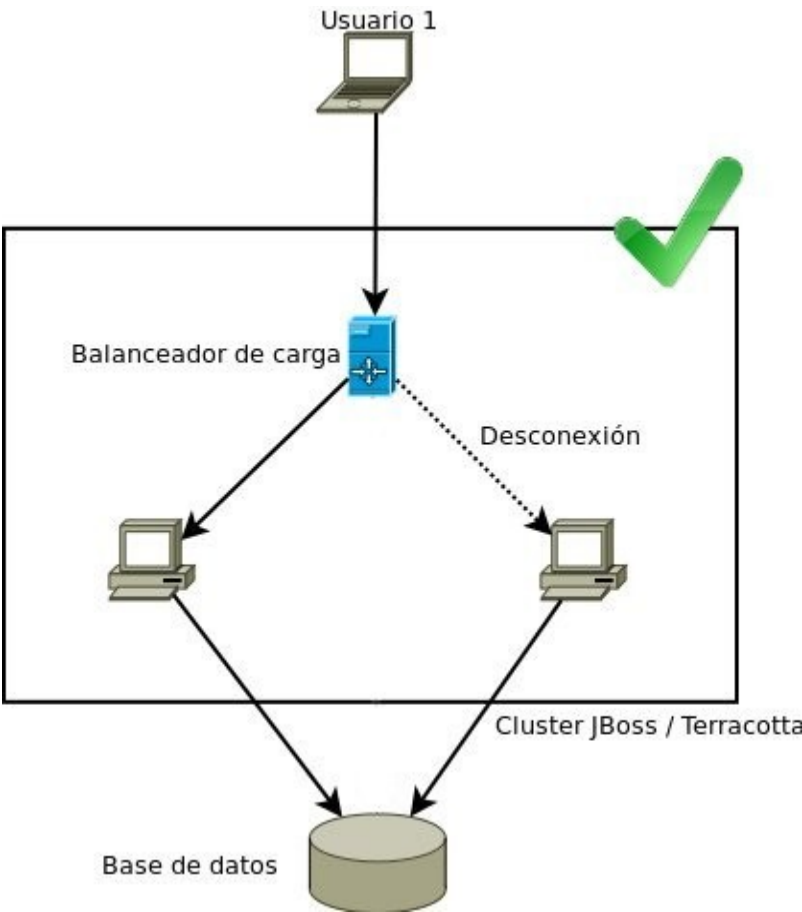
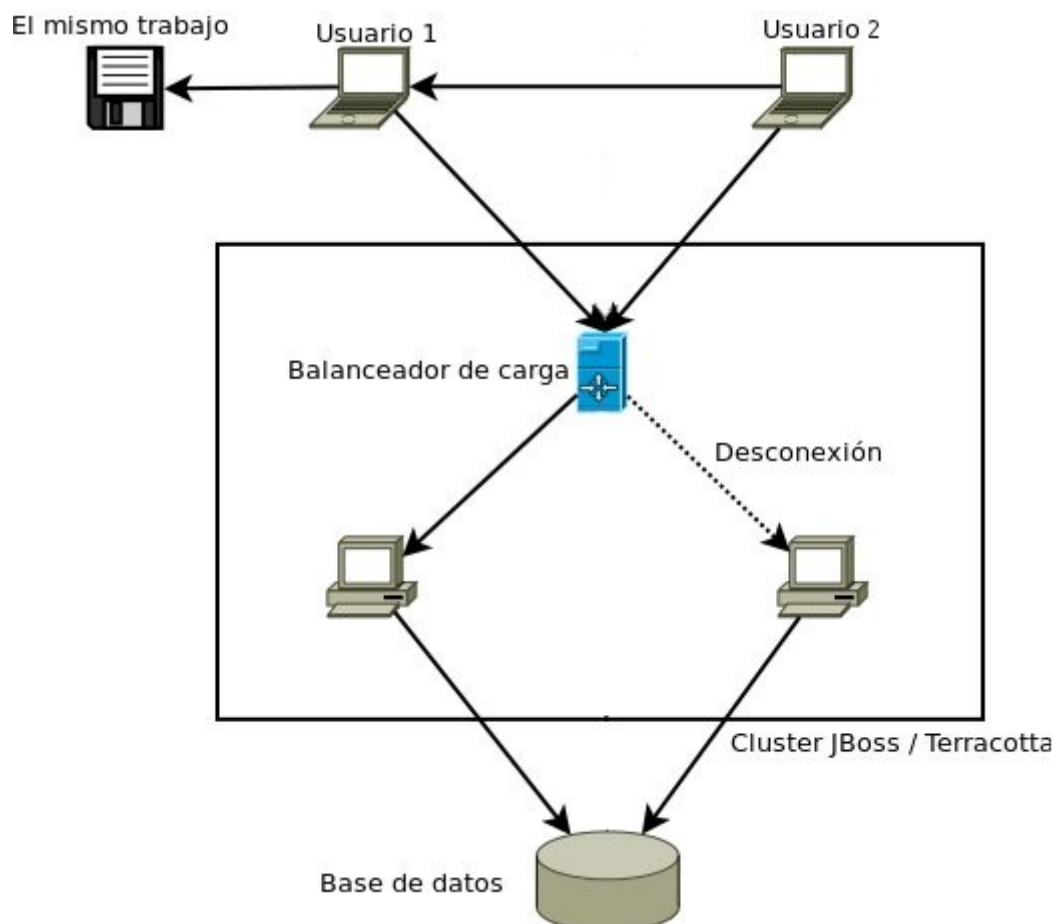


Ilustración 8: Disponibilidad del sistema

Desconexión de un nodo en esquema colaborativo

Descripción		Desconexión de un nodo en un esquema colaborativo, en el momento de confección de resolución colaborativa
Clasificación		Aspecto
		Disponibilidad
Capa de ejecución	VC	X
Tipo de ejecución		Manual
Entrada		Instancia de conexto normal de funcionamiento con texto sincronizado
Salida		Trabajo práctico o dictamen en un determinado estado de sincronización
Criterio de éxito		Correcto funcionamiento e ininterrumpido de la sincronización del texto del trabajo práctico o dictamen

Ilustración 9: Planilla Desconexión de un nodo en esquema colaborativo*Ilustración 10: Desconexión de un nodo en esquema colaborativo*

Replicación de sesión

Descripción		Verificar que ante la desconexión de un nodo, la sesión se replica a los otros nodos y la redirección a otra instancia del servidor es transparente para el usuario.
Clasificación		Aspecto
		Disponibilidad
Capa de ejecución	VC	X
Tipo de ejecución		Manual
Entrada		1-Instancia de escenario normal; ingreso al sistema con un usuario, consulto la pantalla Home para ver mis datos 2- Desconexión del nodo al que ingresamos 3- Nueva consulta de mis datos
Salida		Observar los datos de usuario arrojados en pantalla
Criterio de éxito		Los datos en pantalla indican que se esta apuntando a un nuevo nodo pero manteniendo la misma sesión.

Ilustración 11: Planilla Replicación de sesión

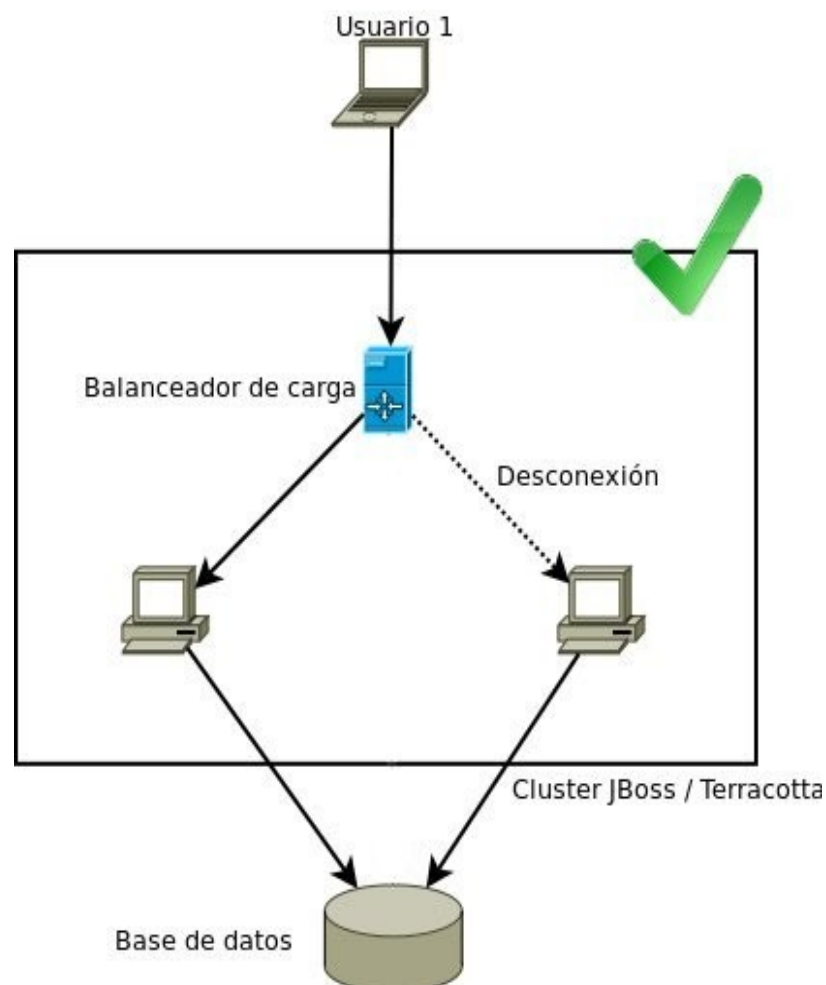


Ilustración 12: Replicación de sesión

Caché

Descripción		Consultar datos cacheables y verificar la replicación de la cache en los diferentes nodos	
Clasificación		Aspecto	
		Distribución de cache	
Capa de ejecución	CN	X	
Tipo de ejecución		Manual	
Entrada		Instancia de escenario normal; ingreso al sistema con un usuario, ingreso en pantalla de búsqueda de trabajos prácticos, ingresar al segundo nodo y consultar la misma pantalla	
Salida		Observar en el log/console de terracotta los hit/miss en la cache	
Criterio de éxito		El segundo nodo continúa administrando la sesión del usuario sin problemas. En la segunda consulta se observan hits en la memoria cache a pesar de estar consultando en el segundo nodo	

Ilustración 13: Planilla Caché

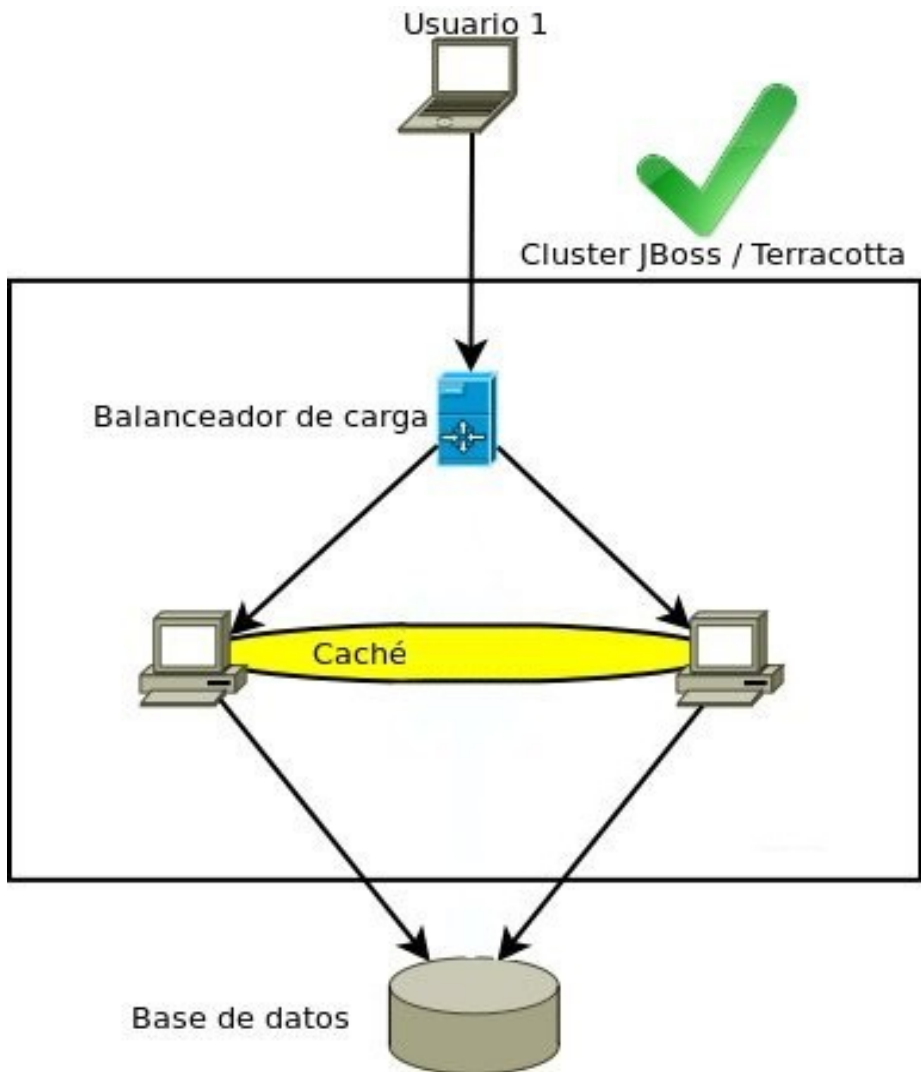


Ilustración 14: Caché

Pruebas automáticas

En nuestro caso implementamos 4 pruebas automáticas que atacan diferentes aspectos del Cluster, y estas pruebas se fueron ejecutando con diferentes cantidades de usuarios, empezando por una cantidad relativamente baja y subiendo de a intervalos, esto lo hicimos para poder observar los tiempos de respuesta y el rendimiento a medida que aumentamos la carga sobre el Cluster.

Las pruebas automáticas implementadas son:

Stress Test (STT)

Descripción		Esta prueba le dá al sistema una carga mucho mayor de la normal o esperada, con el objetivo de observar el comportamiento de la misma ante picos muy altos de utilización.
Clasificación		Aspecto
		Disponibilidad
Capa de ejecución	VC	X
Tipo de ejecución		Automática
Entrada		Escenario de pruebas y plan de pruebas "HttpProxy - Test Plan - Prueba completa"
Salida		Tiempos medios, desviaciones estándar y rendimiento para cada tipo de petición
Criterio de éxito		Que los tiempos medios totales se mantengan dentro de los parámetros establecidos en la tabla de tiempos estándares

Ilustración 15: Planilla Stress Test

El plan de pruebas "HttpProxy - Test Plan - Prueba completa" consiste en:

- Guardar dos resoluciones + sincronizar texto de trabajos prácticos.
- Ver estadística y actualizarla.
- Guardar un Trabajo Practico.
- Guardar dos resoluciones + sincronizar texto de trabajos prácticos.
- Evaluar una resolución.
- Guardar cuatro resoluciones + sincronizar texto de trabajos prácticos.
- Ver estadística y actualizarla.
- Guardar un Trabajo Práctico.

Load Test (LT)

Descripción	Esta prueba le da al sistema una carga normal o esperada, con el objetivo de observar el comportamiento de la misma en situaciones normales de uso.
Clasificación	Aspecto
	Carga
Capa de ejecución VC	X
Tipo de ejecución	Automática
Entrada	Escenario de pruebas y plan de pruebas "HttpProxy - Test Plan - Prueba completa Simple"
Salida	Tiempos medios, desviaciones estándar y rendimiento para cada tipo de petición
Criterio de éxito	Que los tiempos medios totales se mantengan dentro de los parámetros establecidos en la tabla de tiempos estándares

Ilustración 16: Planilla Load Test

El plan de pruebas "HttpProxy - Test Plan - Prueba completa Simple" consiste en:

- Buscar Trabajos Prácticos.
- Iniciar Resolución.
- Sincronizar texto de resolución 10 veces.
- Guardar Resolución.

Load Test Datos Clusterizados (ST)

Descripción	Se prueba específicamente la funcionalidad de sincronización de respuestas, la cual tiene su implementación específica en cada cluster
Clasificación	Aspecto
	Disponibilidad
Capa de ejecución VC	X
Tipo de ejecución	Automática
Entrada	Escenario de pruebas y plan de pruebas "HttpProxy - Test Plan - Prueba de solo sincronizar"
Salida	Tiempos medios, desviaciones estándar y rendimiento para cada tipo de petición
Criterio de éxito	Que los tiempos medios totales se mantengan dentro de los parámetros establecidos en la tabla de tiempos estándares

Ilustración 17: Planilla Load Test Datos Clusterizados

El plan de pruebas "HttpProxy - Test Plan - Prueba de solo sincronizar" consiste en:

- Sincronizar texto de resolución 10 veces.

Load Test Datos No Clusterizados (NST)

Descripción		Prueba de carga normal exceptuando la funcionalidad de sincronización.	
Clasificación		Aspecto	
		Disponibilidad	
Capa de ejecución	VC	X	
Tipo de ejecución		Automática	
Entrada		Escenario de pruebas y plan de pruebas "HttpProxy - Test Plan - Prueba sin sincronizar"	
Salida		Tiempos medios, desviaciones estándar y rendimiento para cada tipo de petición	
Criterio de éxito		Que los tiempos medios totales se mantengan dentro de los parámetros establecidos en la tabla de tiempos estándares	

Ilustración 18: Planilla Load Test Datos No Clusterizados

El plan de pruebas "HttpProxy - Test Plan - Prueba sin sincronizar" consiste en:

- Buscar Trabajos Prácticos.
- Iniciar Resolución.
- Guardar Resolución.

Todas estas pruebas automáticas se pueden mostrar gráficamente de la siguiente forma, donde la "nube" de clientes se logra utilizando Jmeter:

5.6. Problemas encontrados y soluciones propuestas

5.6.1. Generales

El ambiente que armamos primero, sin ningún motivo en particular, fue el de JBoss. Por ende, al ser el primero, el escenario de JBoss nos hizo encontrar numerosos problemas de recursos y configuración en las herramientas en común en los dos ambientes, cuyas soluciones encontradas afortunadamente también influyen en los dos ambientes, a saber:

Recursos en general de las computadoras

Inicialmente, nos planteamos por un tema de prolijidad y organización sobre todo al momento de entregar el trabajo, montar los dos ambientes de Cluster en sendas instancias de virtualización de otro sistema operativo, en este caso en particular el sistema operativo utilizado fue gNewSense Deltah (<http://www.gnewsense.org/>). Dicha elección obedeció a que anteriormente utilizamos gNewSense Deltah en otros contextos, y siempre se había mostrado como uno de los mejores Linux para virtualizar, muy ágil y fluido.

Sin embargo, esta decisión no pudo prosperar, ya que fue imposible para las computadoras Notebook a disposición, levantar la virtualización y sobre ésta última Apache, Terracota + Tomcat ó JBoss.

Finalmente, esta propuesta de presentar el trabajo en una imagen virtualizada fue dejada de lado por un simple tema de recursos de las computadoras. Seguramente es una idea que fomenta la prolijidad, pero lamentablemente no la pudimos implementar.

Los ambientes se montaron, se corrieron y se correrán directamente sobre el sistema operativo primario de las Notebooks.

Repetibilidad

Al notar enseguida que cualquier prueba automática podría ser ejecutada múltiples veces (para anotar mejor los resultados, para poder determinar si el uso de la memoria RAM fue el adecuado, etc), decidimos estructurar los ambientes en scripts de shell. De esta manera, no solo nos pudimos abstraer de recordar continuamente varios detalles del armado de los ambientes, sino que además afrontamos con mayor comodidad la repetición de pruebas, ya que cualquier configuración particular de cualquier ambiente es alcanzada manipulando un script en particular.

Apache

Utilizamos el servidor de aplicaciones Apache (<http://httpd.apache.org/>) versión 2.2 sólo para poder disponer de un balanceador de carga; es decir que su presencia fue meramente auxiliar. El balanceador de carga es vital a nuestros fines, ya que en conjunción con apache, permiten mostrar al Cluster como una única entidad, lo cual es muy provechoso al momento de confeccionar las pruebas; sólo se debe lograr que las pruebas se ejecuten contra el balanceador de carga.

Sin embargo, por supuesto que una vez levantado Apache, se convierte en un actor más en nuestros ambientes, independientemente de que nuestra intención fuera disponer solamente de un balanceador de carga.

En las primeras pruebas con el servidor Apache levantado, debido a una impericia, se nos desarmó el Cluster armado con JBoss; uno de los dos nodos colapsó, y el otro nodo disparó un número muy elevado de peticiones de manera desequilibrada. Posteriormente, notamos que colapsó todo lo que quedaba del Cluster: la base de datos (ver punto siguiente), el Apache, por ende el balanceador de carga, y además el sistema operativo quedó muy lento.

Finalmente, luego de investigar los logs de Apache, JBoss y del sistema operativo en sí, pudimos detectar que el problema se dio por el lado de la configuración de Apache, ya que el nodo de JBoss que emitió muchísimas peticiones de manera desequilibrada, emitió más peticiones que las máximas permitidas por Apache.

Por supuesto que esta situación defectuosa no nos interesa, es decir, no nos interesa que Apache nos permita manejar el desborde de un nodo, siendo que ningún nodo debería desbordar. Sin embargo, como este trabajo no está enfocado en investigar buenas prácticas de uso del Apache, por ejemplo, tomamos la decisión de llevar al máximo la posibilidad de conexiones que tolera el Apache, para poder estar seguros de que cuando todo nos funcionara bien, jamás una prueba se viera interrumpida por el Apache.

Finalmente, editamos en el archivo `/etc/apache2/apache2.conf`, lo siguiente:

```
#
# MaxKeepAliveRequests: The maximum number of requests to allow
# during a persistent connection. Set to 0 to allow an unlimited
# amount.
# We recommend you leave this number high, for maximum
# performance.
#
#MaxKeepAliveRequests 100
MaxKeepAliveRequests 0

...

# prefork MPM
# StartServers: number of server processes to start
# MinSpareServers: minimum number of server processes which are
# kept spare
# MaxSpareServers: maximum number of server processes which are
# kept spare
# MaxClients: maximum number of server processes allowed to start
# MaxRequestsPerChild: maximum number of requests a server process
# serves
#<IfModule mpm_prefork_module>
#   StartServers      5
#   MinSpareServers   5
#   MaxSpareServers   10
#   MaxClients         150
#   MaxRequestsPerChild 0
#</IfModule>
<IfModule mpm_prefork_module>
    StartServers      5
    MinSpareServers   5
    MaxSpareServers   10
    ServerLimit         10000
    MaxClients          10000
    MaxRequestsPerChild 0
</IfModule>

...
```

```
# worker MPM
# StartServers: initial number of server processes to start
# MinSpareThreads: minimum number of worker threads which are kept
spare
# MaxSpareThreads: maximum number of worker threads which are kept
spare
# ThreadLimit: ThreadsPerChild can be changed to this maximum
value during a
#           graceful restart. ThreadLimit can only be changed
by stopping
#           and starting Apache.
# ThreadsPerChild: constant number of worker threads in each
server process
# MaxClients: maximum number of simultaneous client connections
# MaxRequestsPerChild: maximum number of requests a server process
serves
#<IfModule mpm_worker_module>
#   StartServers           2
#   MinSpareThreads       25
#   MaxSpareThreads       75
#   ThreadLimit           64
#   ThreadsPerChild       25
#   MaxClients            150
#   MaxRequestsPerChild   0
#</IfModule>
<IfModule mpm_worker_module>
    StartServers           2
    MinSpareThreads       25
    MaxSpareThreads       75
    ThreadLimit           64
    ThreadsPerChild       25
    ServerLimit          10000
    MaxClients            10000
    MaxRequestsPerChild   0
</IfModule>
```

...

```

# event MPM
# StartServers: initial number of server processes to start
# MinSpareThreads: minimum number of worker threads which are kept
spare
# MaxSpareThreads: maximum number of worker threads which are kept
spare
# ThreadsPerChild: constant number of worker threads in each
server process
# MaxClients: maximum number of simultaneous client connections
# MaxRequestsPerChild: maximum number of requests a server process
serves
#<IfModule mpm_event_module>
#     StartServers          2
#     MinSpareThreads       25
#     MaxSpareThreads       75
#     ThreadLimit            64
#     ThreadsPerChild        25
#     MaxClients            150
#     MaxRequestsPerChild    0
#</IfModule>
<IfModule mpm_event_module>
    StartServers          2
    MinSpareThreads       25
    MaxSpareThreads       75
    ThreadLimit            64
    ThreadsPerChild        25
    ServerLimit            10000
    MaxClients            10000
    MaxRequestsPerChild    0
</IfModule>

```

Siguiendo el mismo razonamiento, llegamos a la conclusión de que tampoco nos era útil que Apache logueara información con un nivel de detalle alto, ya que tampoco queríamos que el costo de loguear en Apache potencialmente atentara contra alguna prueba, por ende en el mismo archivo se editó:

```

...
#
# LogLevel: Control the number of messages logged to the
error_log.
# Possible values include: debug, info, notice, warn, error, crit,
# alert, emerg.
#
LogLevel error
...

```

Y además editamos en `/etc/apache2/sites-available/default:`

```
# Possible values include: debug, info, notice, warn, error, crit,  
# alert, emerg.  
LogLevel error
```

MySQL

En el mismo acontecimiento del punto anterior, también se bloqueó la base de datos, y al intentar conectarse con ella, aparecía el error:

```
java.sql.SQLException: null, message from server: "Host  
'rocha.local' is blocked because of many connection errors;  
unlock with 'mysqladmin flush-hosts'"
```

Esto ocurrió debido a que el nodo que emitió mensajes de manera descontrolada, se comunicó muchísimas veces con el motor de la base de datos, pero como esas comunicaciones fueron siempre incompletas, se sobrepasó el número máximo de conexiones con error que tolera MySQL, por lo que se activó un mecanismo de seguridad por defecto que bajo estas circunstancias, ingresa al servidor que emite las comunicaciones a una lista negra ("black list") en la que se almacenan los servidores con los que MySQL no se va a comunicar.

Para resolver esto, primero se debe ejecutar:

```
mysqladmin -u root -p flush-hosts
```

... y posteriormente, como al igual que en el caso del Apache, nuestra investigación no focaliza sobre MySQL y sus debidos usos, tomamos la decisión de llevar la cantidad máxima de conexiones incorrectas y correctas a niveles muy altos. Editamos en el archivo `/etc/mysql/my.cnf:`

```
#  
# * Fine Tuning  
#  
key_buffer          = 16M  
max_allowed_packet  = 16M  
thread_stack        = 192K  
thread_cache_size   = 8  
# This replaces the startup script and checks MyISAM tables if  
# needed  
# the first time they are touched  
myisam-recover       = BACKUP  
max_connections      = 10000  
max_connect_errors   = 10000  
#table_cache         = 64  
#thread_concurrency  = 10
```


Sistema operativo

En pruebas sucesivas, identificamos en la salida de log de JBoss, el siguiente mensaje:

```
10:07:18,805 WARN  [UDP] send buffer of socket
java.net.DatagramSocket@359ba2ce was set to 640KB, but the OS only
allocated 131.07KB. This might lead to performance problems.
Please set your max send buffer in the OS correctly (e.g.
net.core.wmem_max on Linux)
10:07:18,805 WARN  [UDP] receive buffer of socket
java.net.DatagramSocket@359ba2ce was set to 20MB, but the OS only
allocated 131.07KB. This might lead to performance problems.
Please set your max receive buffer in the OS correctly (e.g.
net.core.rmem_max on Linux)
```

Investigamos esta situación. Dimos con la definición de `wmem_max` y `rmem_max`:

`rmem_max`: ventana máxima de recepción UDP (en bytes).
`wmem_max`: ventana máxima de envío UDP (en bytes).

Es decir que en esta situación, JBoss nos informó que los buffers de envío y recepción UDP estaban configurados muy por debajo de sus máximos. Ambos buffers tenían 128 Kb, y en realidad `rmem_max` puede llegar hasta 24 Mb y `wmem_max` puede llegar hasta 640 Kb

Tal como informa el mensaje de JBoss, esta situación de los buffers tan pequeños podía llevar a problemas de performance, de hecho debajo de la comunicación de Cluster de JBoss está JGroups (<http://www.jgroups.org/>) que es el nombre que se le da a un conjunto particular de herramientas para administrar comunicaciones multicast con UDP (y según configuración también por TCP), por ende seteamos en sus máximos valores a estos parámetros para asegurarnos que no fueran a traer ningún problema.

En el archivo `/etc/sysctl.conf` incorporamos:

```
net.core.wmem_max = 655360
net.core.rmem_max = 26214400
```

... posteriormente ejecutamos `sudo sysctl -p`, y finalmente vimos en consola que los buffers cambiaron de tamaño; a partir de ese momento, JBoss no volvió a informar problemas al respecto.

Y para finalizar los ajustes del sistema operativo, también nos ocurrió que en una de las primeras secuencias exitosas de pruebas, surgieron errores de “too many open files”. Por ende, en el archivo `/etc/security/limits.conf`, al final del mismo, agregamos:

```
* soft nofile 65535
* hard nofile 65535
```

... luego en el archivo `/etc/pam.d/common-session` agregamos:

```
session required                pam_limits.so
```

... ejecutamos `ulimit -n && ulimit -a`, reiniciamos la sesión de usuario, y constatamos que los nuevos límites de cantidad de archivos tomaron lugar.

Generalidades

En un momento, reparamos en el código fuente de la aplicación. Notamos que ciertas operaciones parecían tardar demasiado.

En consecuencia, creamos un Profiler (la clase `com.qin.utils.ProfilingUtils`), que corrimos varias veces junto con la aplicación, para intentar detectar los problemas. Se adjunta una imagen de una de las corridas del Profiler a modo de ejemplo:

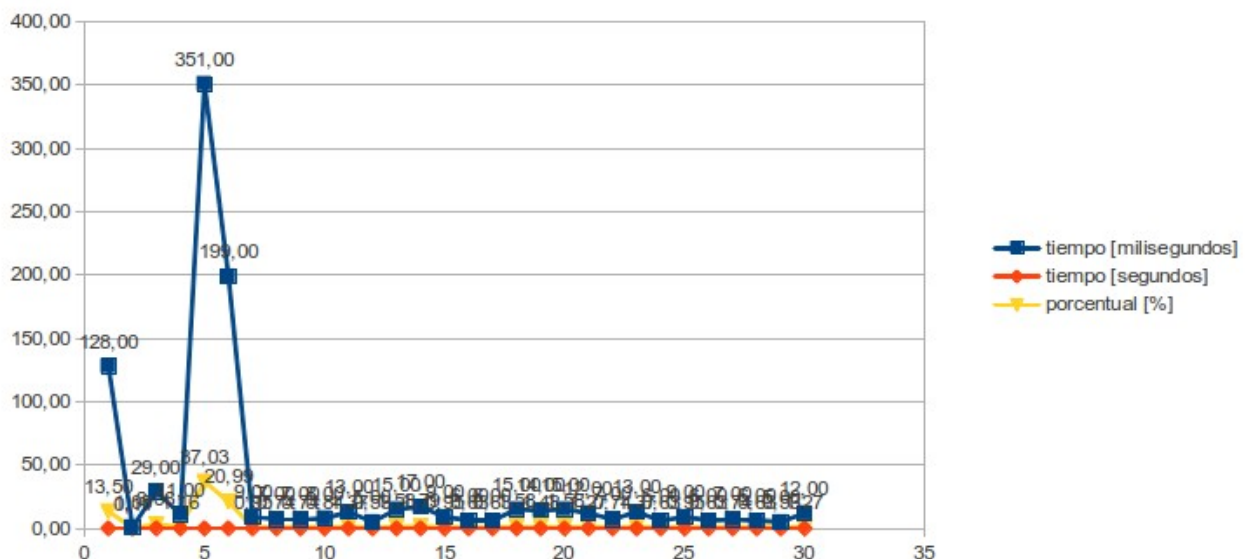


Ilustración 19: Profiler

Lo que se vé aquí, es cómo tarda el procedimiento del alta de TP (punto 5 en el eje X), en comparación a todos los demás procedimientos del sistema; claramente, la rutina de alta de TP necesita un ajuste.

En líneas generales, los ajustes de código fueron:

- Ajustes generales de performance y mejor manejo de excepciones.
- Evitar logear información; que el nivel de log sera fatal o error.
- No utilizar transacciones al hacer consultas de sólo lectura para no trabar a las posibles inserciones, modificaciones o borrados que pudieran darse en simultáneo (ver sección "Transacciones").
- Ingresar al caché las entidades de clase Materia, ya que son invariables; utilizar el cache.
- Variaciones de todo lo anterior, tal como se puede ver en la progresión de commits en <http://code.google.com/p/qin-cluster/source/list>

Transacciones

En las primeras versiones de la aplicación de prueba, todos los métodos de todas las clases de la capa de manager los escribimos con la annotation `@Transactional`. Sin embargo, enseguida encontramos problemas de performance con esquemas de prueba de sólo 300 usuarios, siendo que esperábamos poder trabajar siempre con miles de usuarios.

Luego de experimentar con el código Java y además luego de investigar la situación en Internet, llegamos al diagnóstico adecuado: tal como habíamos configurado los métodos de la capa de manager, con la annotation `@Transactional`, todos los métodos estaban levantando transacciones, incluso aquellos que sólo deben hacer una consulta simple. Entonces, bajo el desgaste de las pruebas, y considerando que lamentablemente nunca pudimos trabajar con servidores web a nivel hardware, la persistencia llegaba a un nivel de saturación tal que trababa el uso de la aplicación.

En consecuencia, tomamos la decisión de utilizar transacciones automáticas en métodos que insertan, editan o borran elementos de la base de datos, de hecho tal como es innegociable, y de no utilizar transacciones bloqueantes en los métodos que sólo realizan consultas.

De esta manera, pudimos solucionar todos los problemas de rendimiento relacionados directamente con el código fuente de la aplicación.

Ver "7.3. Apéndice 3: Detalle del manejo de transacciones".

Java - GC

Mediante prueba y error, se determinó que la configuración Java de Terracotta, Tomcat, y JBoss responda a:

```
-Xms512m -Xmx2048m -Xmn128m -XX:MaxPermSize=256M
```

... a diferencia de JMeter, que tiene esta configuración.

```
-Xms1024m -Xmx1024m -XX:MaxPermSize=256m -XX:NewSize=256m  
-XX:MaxNewSize=256m -XX:PermSize=128m
```

Además, en todos los casos, se utilizan los siguientes parámetros del garbage collector:

```
-Dsun.rmi.dgc.client.gcInterval=3600000 : máximo intervalo de tiempo  
entre llamadas al GC para que vacíe el HEAP local.  
-Dsun.rmi.dgc.server.gcInterval=3600000 : máximo intervalo de tiempo  
entre llamadas al GC para que vacíe el HEAP local.  
-XX:+UseConcMarkSweepGC : habilita el GC concurrente y de baja pausa; la  
aplicación se pausa muy corto tiempo en el recolectado de basura.  
-XX:+CMSClassUnloadingEnabled : recolectar basura de PermGen también  
-XX:+CMSParallelRemarkEnabled : reduce las pausas de remarcado.  
-XX:+UseParNewGC : utilizar todos los núcleos en paralelo para minimizar la  
pausa de recolectado.  
-XX:CMSInitiatingOccupancyFraction=60 : comenzar a correr el thread  
background de recolección cuando esté ocupado más del 60% de la memoria.  
-XX:+UseCMSInitiatingOccupancyOnly : utilizar solamente el parámetro  
anterior para definir cuando levantar el thread del gc.
```

Estos parámetros se utilizan en el entorno productivo de una institución pública de alcance nacional, por lo cual consideramos que es si estos parámetros demuestran rendir en el contexto de dicha institución, entonces seguramente van a rendir en nuestro trabajo.

Servidores

Todos los servidores se levantan con el modo debug en off.

Ver "7.1. Apéndice 1: Cómo utilizar las herramientas seleccionadas".

ACPI

Descubrimos también que todas nuestras computadoras tiene activa la interface ACPI para ajustar el desempeño de la CPU a la exigencia corriente.

En una de las tandas de pruebas, detectamos que la política de cambio de rendimiento de la CPU sobre demanda no nos satisfacía ya que la velocidad en que mensuraba el cambio de demanda era muy pobre, por ende al levantar cualquier componente de nuestros ambientes, corremos esto:

```
echo performance
> /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
echo performance
> /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
echo performance
> /sys/devices/system/cpu/cpu2/cpufreq/scaling_governor
echo performance
> /sys/devices/system/cpu/cpu3/cpufreq/scaling_governor

nucleo0=`more
/sys/devices/system/cpu/cpu0/cpufreq/scaling_governor`
nucleo1=`more
/sys/devices/system/cpu/cpu1/cpufreq/scaling_governor`
nucleo2=`more
/sys/devices/system/cpu/cpu2/cpufreq/scaling_governor`
nucleo3=`more
/sys/devices/system/cpu/cpu3/cpufreq/scaling_governor`
```

... que cambia la política de sobre demanda, por la política de siempre tener la CPU en su máximo rendimiento.

Ver "7.2. Apéndice 2: Breve descripción de los scripts creados".

Balanceador de carga

Inicialmente comenzamos utilizando mod_jk, pero con el paso del tiempo, nos dimos cuenta de que nos era mucho más práctico utilizar mod_proxy, ya que el mismo provee una funcionalidad llamada balancer-manager, que permite controlar mediante un navegador web el status del Cluster. Fue muy práctico para efectuar seguimiento del comportamiento del Cluster.

localhost/balancer-manager

Load Balancer Manager for localhost

Server Version: Apache/2.2.22 (Ubuntu) mod_jk/1.2.32
Server Built: Feb 13 2012 01:37:45

LoadBalancer Status for balancer://ajpcluster

StickySession Timeout FailoverAttempts Method
JSESSIONID 0 1 byrequests

Worker URL	Route	RouteRedir	Factor	Set	Status	Elected	To	From
ajp://worker1:8009	worker1		1	0	Ok	0	0	0
ajp://worker2:8109	worker2		1	0	Ok	0	0	0

Apache/2.2.22 (Ubuntu) Server at localhost Port 80

Ilustración 20: Balancer Manager

Cache

Una herramienta frecuentemente utilizada para mejorar la performance de una aplicación es la cache. En nuestro caso decidimos implementar una cache de 2do nivel, esto significa que las entidades puestas en la cache se mantienen inclusive entre diferentes transacciones con la base de datos.

Esta cache la utilizamos con varias entidades, siempre con el objetivo en mente de obtener mayor performance de la que estábamos midiendo; Sin embargo una de las entidades cacheadas fue la que representa al Trabajo Práctico, y una de las características de esta entidad es que sufre altas, bajas y modificaciones con frecuencia relativamente alta. Al hacer esto encontramos que existe una gran diferencia entre las cache locales y las clusterizadas, ya que al clusterizar una cache, la misma se replica en cada nodo del Cluster, lo que significa que cada miss representa no solo actualizar una cache, sino que implica sincronizar el resto de las caches del Cluster, lo que finalmente se traduce en un gran overhead.

Por otro lado dependiendo del tipo de cache también se pueden introducir bloqueos de threads gracias a que la cache no devuelve información hasta asegurarse de estar actualizada (o sea, solo devuelve información luego de sincronizar con el resto de los nodos) consiguiendo una disminución de la performance muy significativa.

En conclusión, se debe prestar especial atención a la hora de elegir las entidades/queries cacheables, en especial al trabajar con caches clusterizadas, ya que se pueden introducir grandes disminuciones en la performance.

Distribución de sesiones

Para poder comprobar que las sesiones de usuario se distribuyen correctamente, decidimos aplicar una prueba que se propone en uno de los ejemplos del paquete Terracotta. La misma consiste en:

- Logearse en la instancia del sistema desplegado en un nodo cualquiera, sin utilizar el balanceador de carga.
- Efectuar algunas operaciones básicas del sistema.
- Cambiar la URL, y apuntar al otro nodo, sin pasar por el balanceador de carga.
- Conclusión: debe verse que se actualiza la URL indicando que se cambió de nodo, que el usuario está también logeado en el segundo nodo, y que todas sus operaciones están tal cual estaban en el nodo original.

Si esto se cumple, se concluye que la sesión está correctamente distribuida.

Inicialmente, esta prueba no la pudimos comprobar, ya que en el tercer paso al cambiar la URL, el segundo nodo nos bloqueaba el acceso, o nos cambiaba la sesión de usuario.

Luego de experimentar lo necesario, llegamos a la conclusión de que el inconveniente que encontramos lo genera el navegador de internet: cuando detecta que el IP cambia, exige la creación de otra sesión de usuario, lo cual colisiona con la propuesta que le hace el nodo de seguir trabajando con la sesión original.

Para sortear este inconveniente, modificamos la prueba de la siguiente manera:

- Agregar una entrada en `/etc/hosts` que sirva de etiqueta para invocar al nodo con que vamos a comenzar la prueba.
- Logearse en el sistema desplegado en el nodo cuya etiqueta creamos en el punto anterior, sin utilizar el balanceador de carga, apuntando con el navegador de internet directamente a la etiqueta creada.
- Efectuar algunas operaciones básicas del sistema.
- Editar la entrada creada anteriormente en `/etc/hosts`, y hacer que ahora refiera al otro nodo que forma parte del cluster
- Refrescar la URL escrita en el navegador de internet; ahora el mismo IP para el navegador, direcciona a otro nodo del cluster, por lo cual al refrescar la pantalla se cambia de nodo sin que el navegador de internet lo perciba y sin que el mismo gestione una nueva sesión.
- De esta manera, ocurre tal como se comenta en el ejemplo de Terracotta, que el usuario está también logeado en el segundo nodo, y que todas sus operaciones están tal cual estaban en el nodo original.

Esto lo pudimos comprobar en ambos clusters, por lo cual concluimos que efectivamente las sesiones de usuario se distribuyen correctamente.

5.6.2. JBoss

Cluster vertical

En etapa de desarrollo, resultó muy práctico poder levantar todo el Cluster en una sola computadora, para poder comprobar que los pasos a seguir fueran correctos.

Por ende, surgió el interrogante de cómo poder disponer de múltiples instancias de JBoss en la misma computadora. Dicho interrogante se resolvió al leer [JBoss AS IN ACTION], donde se explica como trabajar para que no haya colisión de archivos ni de puertos al utilizar múltiples instancias de JBoss en la misma computadora.

Finalmente, la combinación adecuada es instalar JBoss en tantos directorios distintos como instancias se quiera tener, levantar todas esas instancias utilizando el perfil all, y utilizar ports-default en una de las instancias, y correr los puertos de a cien unidades con ports-XY en las demás instancias.

Ver "7.1. Apéndice 1: Cómo utilizar las herramientas seleccionadas".

Hornetq

En los primeros experimentos con el Cluster de JBoss, notamos que algunas veces, luego de bajar el Cluster y posteriormente volver a levantarlo, la comunicación intra - Cluster no se podía dar, y aparecían errores de hornetq.

Pudimos comprobar, investigación mediante, que cuando esto es así, se debe a que en la última comunicación exitosa, se efectuó una serie de serializaciones en los directorios `server/all/data/hornetq` de cada instancia de JBoss afectada, que posteriormente se alteró y dejó de funcionar. Parece ser un bug de JBoss 6.1.0.Final, ya que con solamente borrar los contenidos de los directorios `server/all/data/hornetq` de las instancias afectadas, en el siguiente intento la comunicación se da de manera perfecta.

El borrado preventivo de dichos directorios fue incorporado a los scripts de trabajo.

Ver "7.2. Apéndice 2: Breve descripción de los scripts creados".

Singleton

Como se informó anteriormente, en la aplicación de pruebas hay una funcionalidad de sincronización que debe ser única en todo el Cluster.

En el Cluster de JBoss, inicialmente se implementó dicha funcionalidad con la annotation `@Singleton`, porque nos pareció la opción evidente y trivial.

Sin embargo, estábamos totalmente equivocados. Con dicha annotation se puede generar un singleton a nivel servidor, no llega a estar distribuido en todo el Cluster, solo tiene injerencia en el nodo en que se define, por lo cual si bien nuestro objetivo era tener una funcionalidad que sincronice a nivel Cluster, conseguimos inicialmente una funcionalidad por nodo que sincroniza a nivel nodo, lo cual no nos sirvió.

La solución a este inconveniente se ve más en detalle en la sección que sigue.

Replicación de datos centralizados

La funcionalidad de resolución colaborativa de trabajos prácticos requiere de compartir el texto de las resoluciones entre los usuarios que pueden estar conectados a diferentes nodos del Cluster. Para atacar este problema decidimos implementar una caché distribuida de entidades no persistentes.

Investigando esta posibilidad, dimos con un problema que a partir de que lo supimos detectar, lo pudimos reconocer en todos y cada uno de los inconvenientes vividos. Se intenta explicar a continuación sus aspectos más importantes:

- Al comenzar este trabajo, elegimos utilizar el servidor JBoss cuya versión se puede ver en "7.1. *Apéndice 1: Cómo utilizar las herramientas seleccionadas*"; las versiones posteriores del mismo en aquel momento nos parecieron demasiado nuevas como para ser suficientemente estables, y las versiones anteriores al mismo parecieron estar suficientemente superadas como para elegir las.
- La documentación de las versiones previas a la elegida, es realmente muy completa y útil; además, dichas versiones previas tenían integradas las herramientas de JBossCache, y PojoCache (<http://www.jboss.org/jboss-cache>), dos herramientas que nos interesaba sumar, pero que finalmente son complicadas de utilizar debido a lo que se explica a continuación.
- La documentación de la versión elegida de JBoss quedó inconclusa, porque cuando la comunidad de JBoss resolvió a dar origen a la versión posterior, tomó la decisión de prácticamente eliminar todo el código fuente y empezar nuevamente, por ende toda la documentación de la

versión elegida se torna obsoleta automáticamente, por lo cual ante la evidencia de un esfuerzo sin sentido cesó el proceso de documentación de la versión de JBoss elegido.

- Como la versión nueva de JBoss es una reescritura por completo, su documentación no puede ser tomada para nuestro caso.
- Los componentes de JBossCache y PojoCache tan correctamente integrados y documentados en las versiones previas, en la versión elegida han sido reemplazados por Infinispan, cuya documentación no es suficiente y además adolece de un reemplazante natural para PojoCache.

Teniendo todo lo anteriormente nombrado en cuenta, se decidió:

- Leer la documentación de todas las versiones posibles y sacar de esa lectura todas las conclusiones que se puedan sacar, y realizar todas las pruebas y todos los códigos de prueba que se puedan hacer.
- Utilizar Infinispan sólo para cache de segundo nivel de persistencia.
- Integrar manualmente a las instancias de JBoss, el componente PojoCache.
- No trabajar con JbossCache.

Ver sección "Cache de segundo nivel" de este punto, y ver "7.4.

Apéndice 4: Diferencias en la implementación de la sincronización" y "7.5.

Apéndice 5: Código de la clase PojoCacheManagerImpl, utilizada en el Cluster con JBoss".

Cache de segundo nivel

En el Cluster con JBoss se implementó una cache de segundo nivel, con la utilidad que trae por defecto JBoss 6 para tales fines, Infinispan (<https://www.jboss.org/infinispan>).

Todo el proceso para lograr hacer andar una cache de segundo nivel en JBoss fue bastante sufrido. A la inexperiencia, que siempre tiene un costo, se suma una situación por demás particular con la documentación y con la librería en sí, explicado en el punto anterior y repasado a continuación.

En lo que respecta a documentación de caches de segundo nivel en JBoss, cabe destacar que la documentación más rica es la de JBoss 5, pero en esa versión, se usaba JBossCache. En nuestro caso, estamos usando JBoss 6. Nos encontramos con que no solamente la documentación no es tan buena para JBoss 6, sino que además en JBoss 6 JBossCache fue reemplazado por Infinispan. Para completar el cuadro, no se puede obtener ayuda desde la documentación de JBoss 7 porque JBoss 7 fue prácticamente reescrito.

De todas maneras, luego de unas intensas sesiones de investigación, prueba y error, se logró con éxito activar Infinispan en JBoss 6 como cache de segundo nivel.

Los resultados que obtuvimos, distaron de ser los esperados y los deseados. Si bien dejamos este cache activo, hemos visto como al estar activo se siente que el desempeño del Cluster con JBoss en su totalidad es peor que el desempeño del mismo Cluster sin este cache de segundo nivel.

Creemos que esto se debe a:

- Las características de uso no configurables que tiene este cache: en la documentación [INFINISPAN] se ve con claridad que al entrar un leer un objeto de la base de datos, necesariamente este último se almacena en el cache local, para evitar comunicaciones intra - Cluster de más. Además, cuando una entidad es actualizada en un cache, se dispara un mensaje a todos los otros caches del Cluster informando que se deben invalidar y refrescar por completo. Ninguna de estas características es configurable.
- Lo que se expresa claramente en la documentación [INFINISPAN]: se cita textualmente.

"On Caching

Query result caching, or entity caching, **may or may not improve** application performance. Be sure to benchmark your application with and without caching."

Claramente, este es un caso en que no se mejora la performance, sin lugar a dudas.

Replicación de sesiones

Por inexperiencia, a priori ignorábamos cómo marcar las sesiones de JBoss como distribuidas. Por suerte esto fue exageradamente más simple que en el caso anterior, al marcar el web.xml como `<distributable/>` se logra esto automáticamente.

5.6.3. Terracotta + Tomcat

Replicación de datos centralizados

En terracotta existen dos formas para poder compartir los datos de las resoluciones de trabajos prácticos inclusive entre los distintos nodos del Cluster.

1.- La primera es totalmente por configuración, con lo que se hace uso de las capacidades de terracotta de emular una única JVM a partir de las JVM de los nodos, de esta forma cualquier variable dentro de java (que se encuentre correctamente configurada) puede ser compartida entre todos los nodos del Cluster, logrando así compartir datos sin necesitar de soluciones adicionales. Sin embargo este enfoque no es el recomendado en la documentación de Terracotta, ya que requiere tener en cuenta y configurar "a mano" bloqueos para evitar problemas de concurrencia. La documentación solo recomienda esta opción ante situaciones en las que no se pueda utilizar la opción número 2 [TERRACOTTA VENTAJAS TOOLKIT]

2.- La segunda opción, la recomendada en la documentación (y la que utilizamos en nuestro sistema), es mas intrusiva porque requiere de una modificación en el código fuente de la aplicación. En nuestro caso necesitamos usar una estructura tipo Mapa para contener el estado de una Resolución compartida en progreso, y teniendo en cuenta la recomendación de Terracotta el único cambio a realizar es cambiar el mapa utilizado por uno de terracotta [TERRACOTTA TOOLKIT]:

```
Map mapaResoluciones = new  
TerracottaClient("ip_terracotta:9510").getToolkit().getMap("myStaticMap");
```

De esta forma, evitamos todo tipo de configuración, y todo el manejo de concurrencia y sincronización del mapa está dado automáticamente.

Cache de segundo nivel

Utilizada de forma correcta y teniendo en cuenta los problemas presentados anteriormente, se puede mejorar la performance del sistema. Terracotta esta preparado para utilizar la librería EhCache [TERRACOTTA EHCACHE] , que es precisamente la librería que utiliza el framework de persistencia utilizado (Hibernate).

La configuración necesaria para habilitar dicha cache se realiza en dos pasos:

1- Archivo hibernate.cfg.xml:

```
<property  
name="net.sf.ehcache.configurationResourceName">ehcache.xml</prope  
rty>
```

2- Archivo de configuración ehcache.xml:

```
<ehcache name="QinWebCache">  
  <defaultCache maxElementsInMemory="15000" eternal="false"  
    timeToIdleSeconds="120" timeToLiveSeconds="120"  
    overflowToDisk="false">  
    <terracotta />  
  </defaultCache>  
  <terracottaConfig url="ip_terracotta:9510" />  
</ehcache>
```

Replicación de sesiones

Terracotta soporta la replicación de sesiones entre los diferentes nodos del Cluster [TERRACOTTA SESSIONS]. Esto se puede lograr fácilmente, modificando el archivo *context.xml* de de Tomcat:

```
<Valve  
className="org.terracotta.session.TerracottaTomcat70xSessionValve"  
tcConfigUrl="ip_terracotta:9510"/>
```

Gracias a esta configuración la sesión del usuario se replica a los diferentes nodos, de manera que ante cualquier falla de un servidor, el usuario no experimenta ningún inconveniente.

Array de Servidores Terracotta

Por último un problema que aparece con la versión gratuita de Terracotta, es que para mantener todos los datos clusterizados (sesiones, cache, y datos particulares) se necesita de un servidor extra donde se ejecuta el "Servidor Terracotta" (herramienta encargada de todo el manejo de sincronización, etc), y dado que la versión gratuita solo permite tener uno de estos servidores, éste se convierte en un punto de posible falla muy importante.

Este problema se soluciona con la versión paga de la herramienta, ya que permite clusterizar no solo el sistema en cuestión, sino también el "Servidor Terracotta", teniendo así redundancia y evitando posibles fallos de la herramienta en si.

5.7. Resultados

Como explicamos en la sección de 5.5. Pruebas, utilizamos Apache JMeter para realizar las pruebas automáticas. Esta herramienta arroja resultados de la siguiente forma:

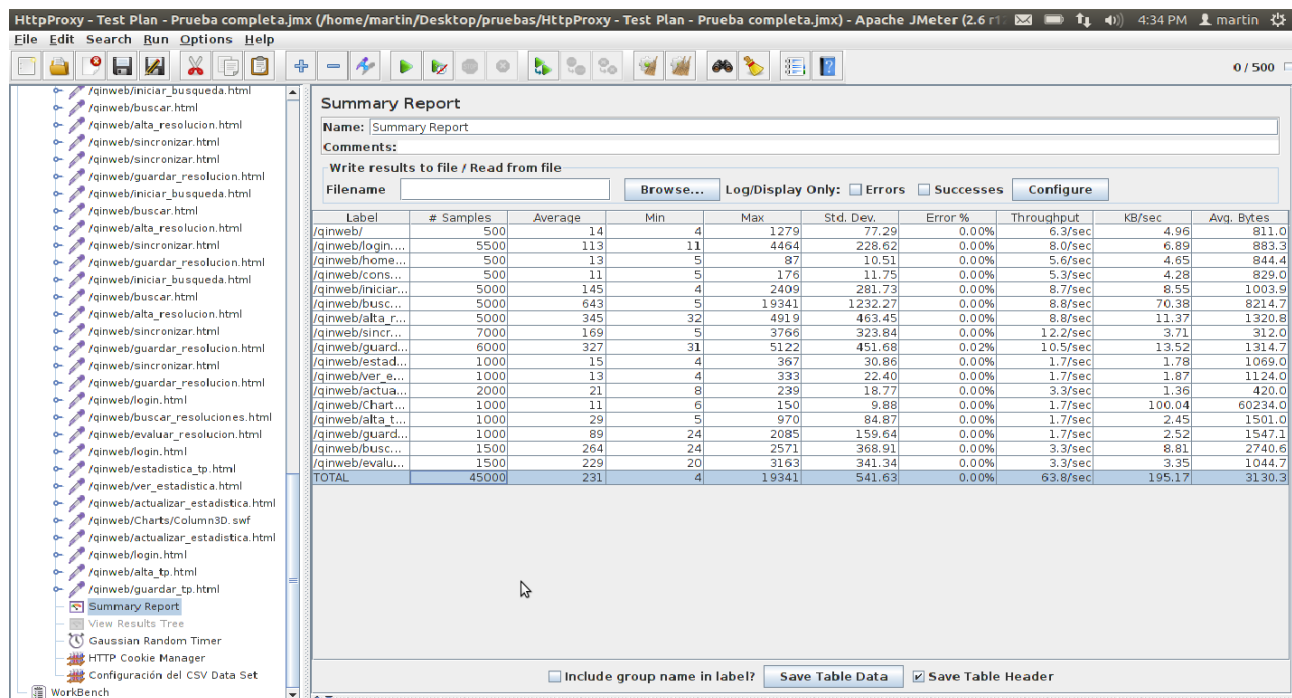


Ilustración 21: Resultados en JMeter

Podemos ver a la derecha las peticiones que se realizan al sistema, y a la izquierda en forma de tabla podemos observar la cantidad de peticiones de cada tipo, los tiempos medios de respuesta, las desviaciones estándares, porcentaje de error, rendimiento y mas.

En todos los casos:

- Rendimiento: [resquests/segundo].
- Media: [milisegundos].
- Desviación estándar: [milisegundos].
- Eje X: cantidad de usuarios.

- Para determinar si las mediciones están dentro de los tiempos estándar, se debe recordar la tabla en "5.5. Pruebas". Sintetizando, la información contenida en dicha tabla indica que: la aplicación responde correctamente si cualquier petición tarda entre 0,1 y 10 segundos. Una respuesta de 1 segundo es el punto medio de dicho intervalo. Por ende, para poder concluir a partir de los gráficos que la aplicación se encuentra dentro de su comportamiento estándar, considerando que en dichos gráficos hay promedios acumulados en las distintas corridas con JMeter, se define que media + la desviación estándar en cualquier punto debe ser como mucho igual a un 1 segundo; cuando esto se cumpla, la aplicación en ese punto presentará los valores estándar de respuesta. Llamaremos a esta medición, por motivos de practicidad, "punto de tiempo estándar".

5.7.1. JBoss

Pruebas manuales

Las pruebas manuales fueron muy prácticas en la etapa de desarrollo, ya que sirvieron para poder comprobar paso a paso que los Clusters respondieran como es debido.

Todos los casos de prueba manuales fueron ejecutados con éxito, con lo que pudimos seguir con pruebas mas exigentes, tanto manuales como automáticas.

Como muestra se presenta el siguiente caso, el cual es una combinación de los casos de prueba manuales *"Desconexión de un nodo en esquema colaborativo"* y *"Caché"* presentados en la sección "5.5. Pruebas":

- Levantar el Cluster.
- Ingresar con dos usuarios distintos, en dos navegadores distintos.
- Repetir el punto anterior hasta que un usuario sea atendido por un nodo del Cluster, y el otro usuario sea atendido por el otro nodo del Cluster.
- Ir con ambos usuarios, a resolver un mismo trabajo práctico de manera colaborativa.
- Escribir texto para que se sincronice.
- Bajar el nodo que se levantó primero.
- Confirmar que la aplicación sigue andando.
- Confirmar que el usuario que fue atendido por el nodo bajado, ahora es atendido por el otro nodo; el cambio se produce de manera transparente para el usuario.
- Volver a levantar el nodo que se bajó anteriormente.
- Comprobar que el Cluster se vuelve a armar.
- Comprobar mediante la consola administrativa del servidor de sincronización, que el caché ha sido usado.

Habiendo ejecutado la prueba anterior correctamente, ya no queda duda alguna del correcto funcionamiento del Cluster, por lo que a partir de ese momento nos pudimos enfocar en las pruebas automáticas para obtener información relacionada con tiempos de respuesta y rendimiento. Para información detallada de la salida de dicha prueba ver "7.6. Apéndice 6: Ejecución de la combinación de casos de pruebas manuales".

Pruebas automáticas

Stress Test (STT) - Cluster

Total hilos	Rendimiento total	Media total datos clusterizados	Media total datos sin clusterizar	Desviación estándar total datos clusterizados	Desviación estándar total datos sin clusterizar
300	41,30	15,33	17,31	7,74	7,95
600	81,68	20,33	17,83	28,44	13,61
900	119,68	20,33	23,75	15,76	14,36
1100	145,31	27,67	30,35	27,42	23,03
1200	155,78	43,33	46,35	53,48	52,28
1300	170,62	65,67	64,50	70,74	77,80
1400	182,30	164,00	141,15	310,25	232,44

STT - JBoss - Cluster

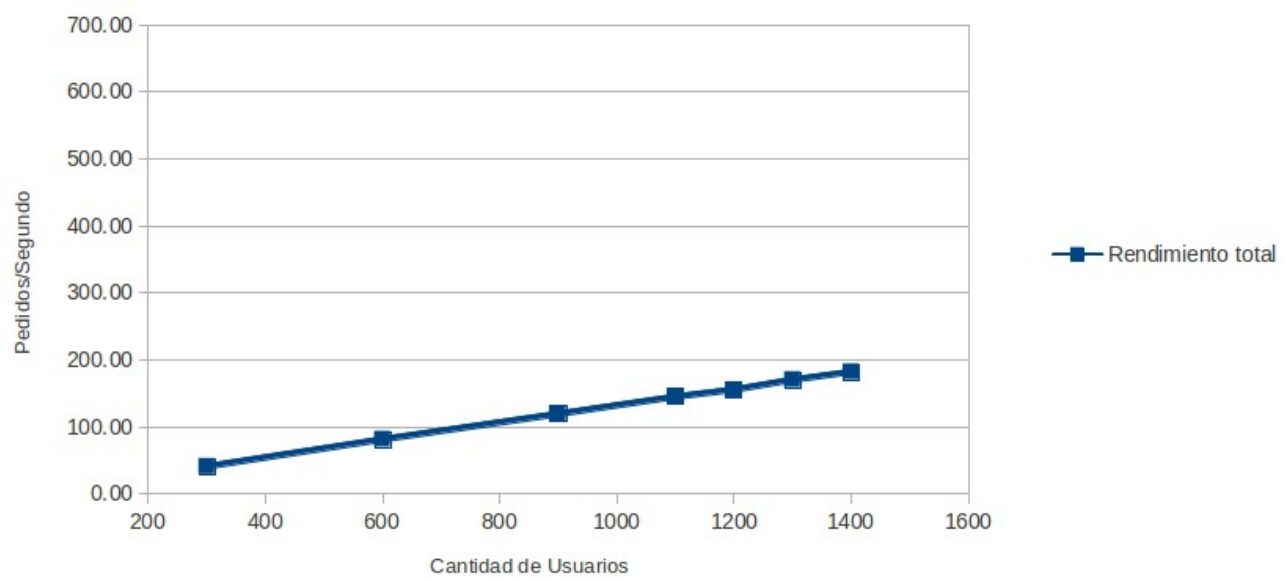


Ilustración 22: STT - JBoss - Cluster

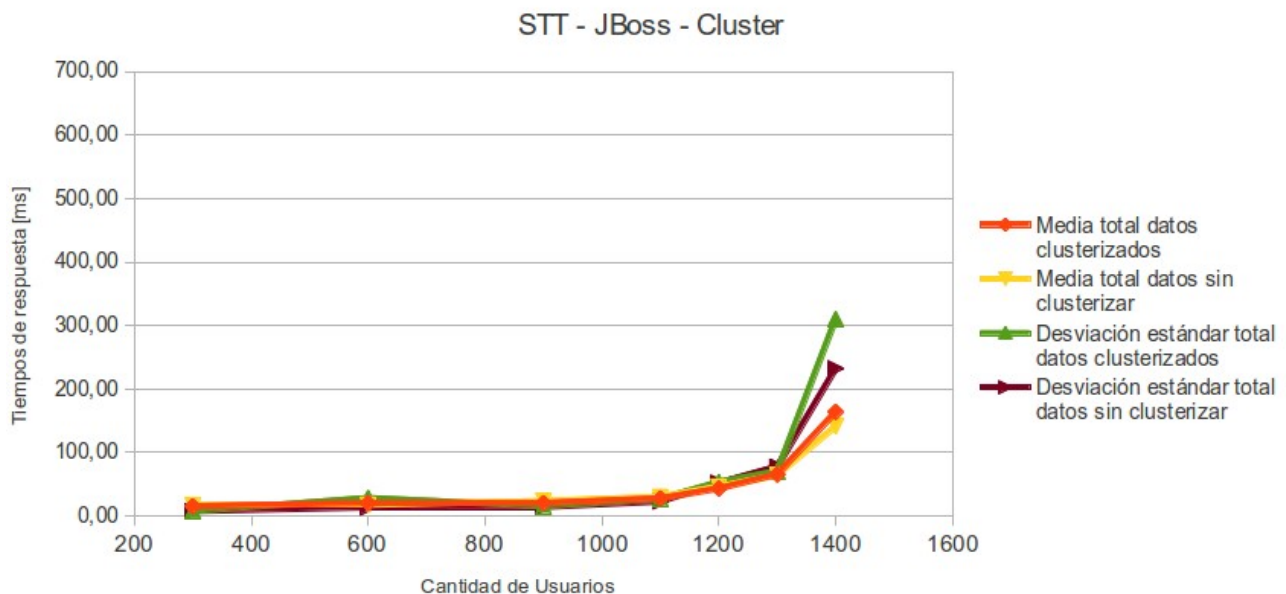


Ilustración 23: STT - JBoss - Cluster

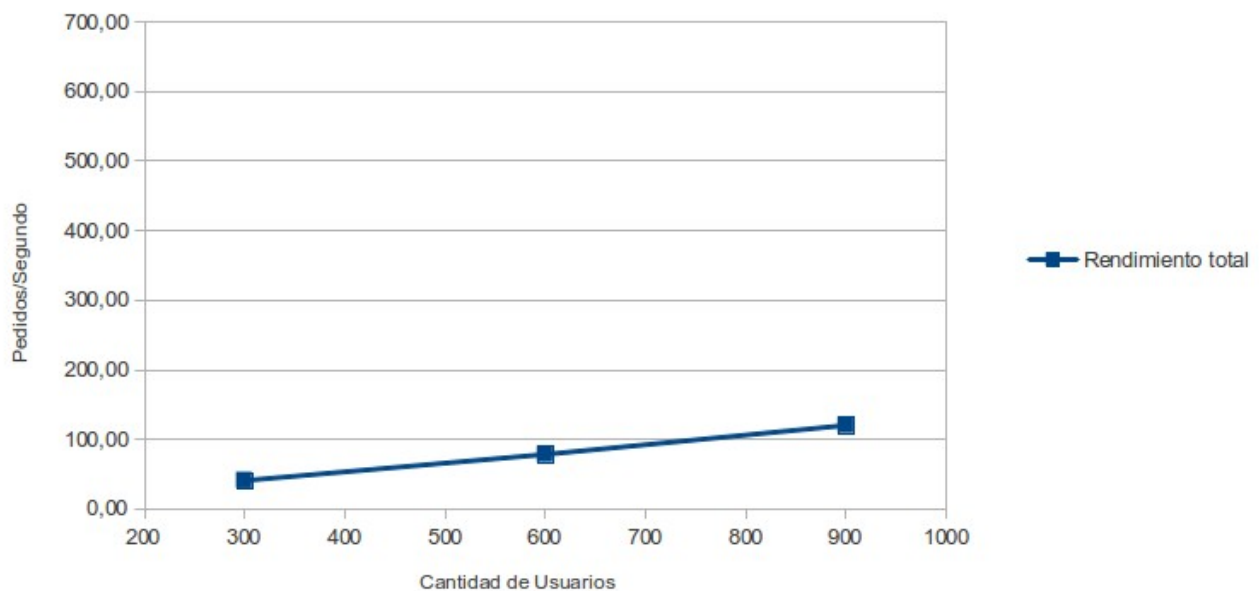
El punto de tiempo estándar más alto en toda la prueba, es de 474,25 milisegundos, para los datos clusterizados con 1400 usuarios, que es un valor aceptable. Se concluye que en toda la prueba, los tiempos fueron los deseables.

Como se puede ver, hasta los 1300 usuarios (en dos nodos; podríamos decir como máximo 650 usuarios por nodo en una foto ideal del funcionamiento en equilibrio) el comportamiento de la aplicación es el deseable. Desde los 1300 usuarios en adelante, el Cluster con JBoss se enfrenta a la escasez de recursos, propia de trabajar un escenario tan exigente como este, sobre una notebook, y no sobre un servidor especialmente preparado para esto.

En esta prueba, el comportamiento del Cluster de JBoss parece ser el adecuado.

Stress Test (STT) - Individual

Total hilos	Rendimiento total	Media total datos clusterizados	Media total datos sin clusterizar	Desviación estándar total datos clusterizados	Desviación estándar total datos sin clusterizar
300	41,02	5,33	14,04	3,99	8,77
600	78,58	4,33	14,27	4,32	8,38
900	120,33	5,33	19,48	9,22	19,46

STT - JBoss - Individual*Ilustración 24: STT - JBoss - Individual*

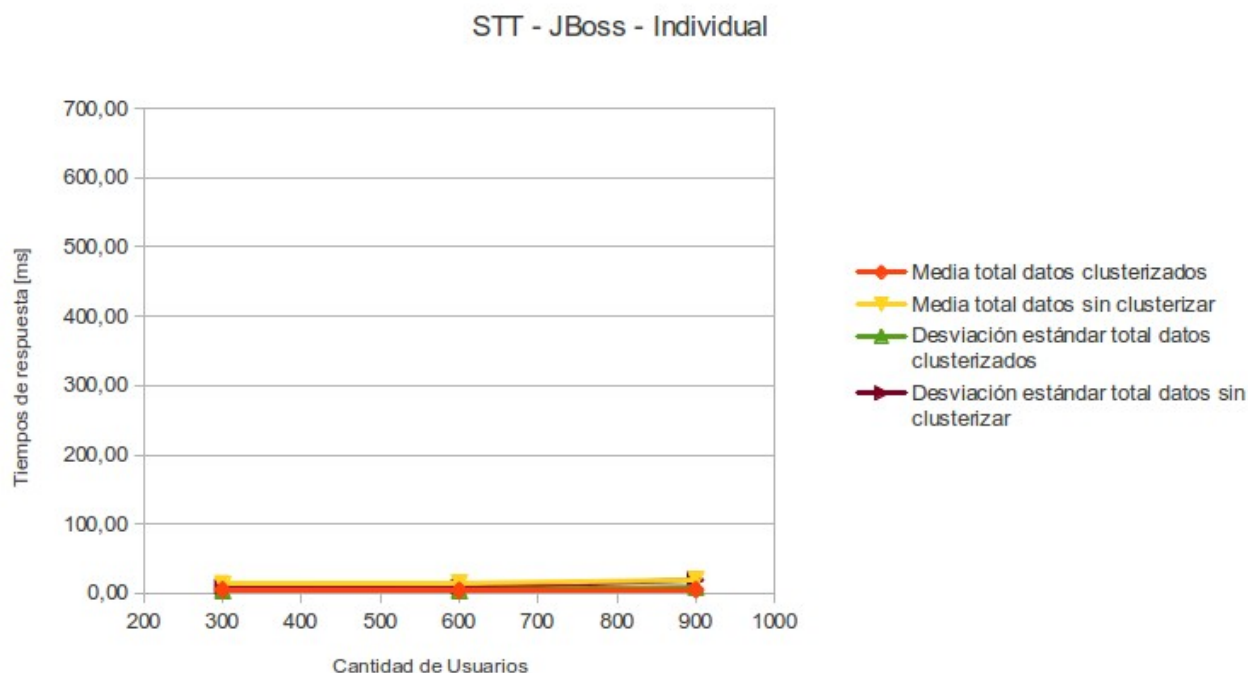


Ilustración 25: STT - JBoss - Individual

El punto de tiempo estándar más alto en toda la prueba, es de 38,94 milisegundos, para los datos sin clusterizar con 900 usuarios, que es un valor aceptable. Se concluye que en toda la prueba, los tiempos fueron los deseables.

En este caso, se puede ver que el nodo individual es mucho más exigido que en la configuración en Cluster; en la oportunidad anterior, se pudieron atender a 1300 usuarios, en este caso, con un nodo individual, se pueden atender solamente a 900, pero con la particularidad de que los 900 van al mismo nodo, frente a la foto ideal en equilibrio del caso anterior de como máximo 650 usuarios por nodo.

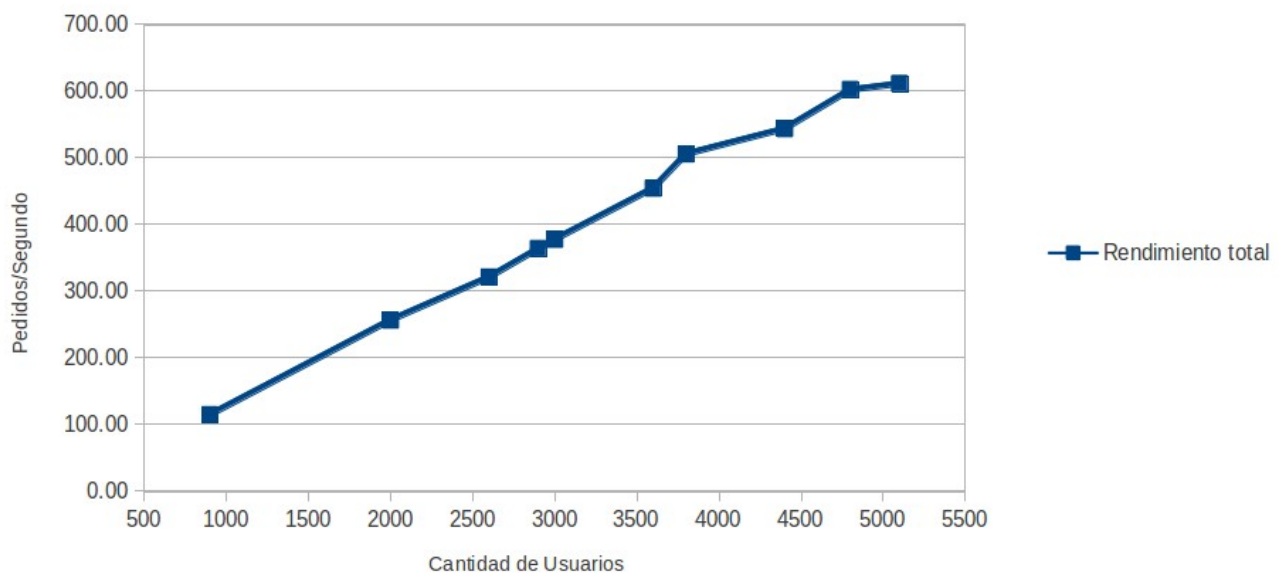
Se puede apreciar aquí una particularidad de la comparación nodo individual - Cluster: mientras responde el nodo individual, es más rápido que el Cluster lo cual también se puede concluir comparando ambos puntos de tiempo estándar, sin embargo, el Cluster tiene más escalabilidad y por definición brinda alta disponibilidad.

Otra particularidad detectada, es que cuando se trabaja con el Cluster, el punto de tiempo estándar tiene su pico máximo con los datos clusterizados, lo cual permite concluir que evidentemente hay un costo en la sincronización de esos datos, que en el mismo escenario no se ve con los datos sin clusterizar.

Load Test (LT) - Cluster

Total hilos	Rendimiento total	Media total datos clusterizados	Media total datos sin clusterizar	Desviación estándar total datos clusterizados	Desviación estándar total datos sin clusterizar
900	114,88	8,00	18,50	4,76	6,25
2000	256,82	17,00	23,46	17,59	15,36
2600	321,41	19,67	29,54	22,02	25,86
2900	364,00	22,67	35,29	24,98	27,05
3000	377,91	23,67	32,54	29,91	28,39
3600	454,70	47,00	59,83	63,32	61,00
3800	506,02	65,67	85,75	81,93	84,90
4400	543,98	97,00	111,17	137,07	129,10
4800	602,43	189,00	178,42	228,64	186,55
5100	611.43	252.70	243.92	368.74	335.85

LT - JBoss - Cluster

*Ilustración 26: LT - JBoss - Cluster*

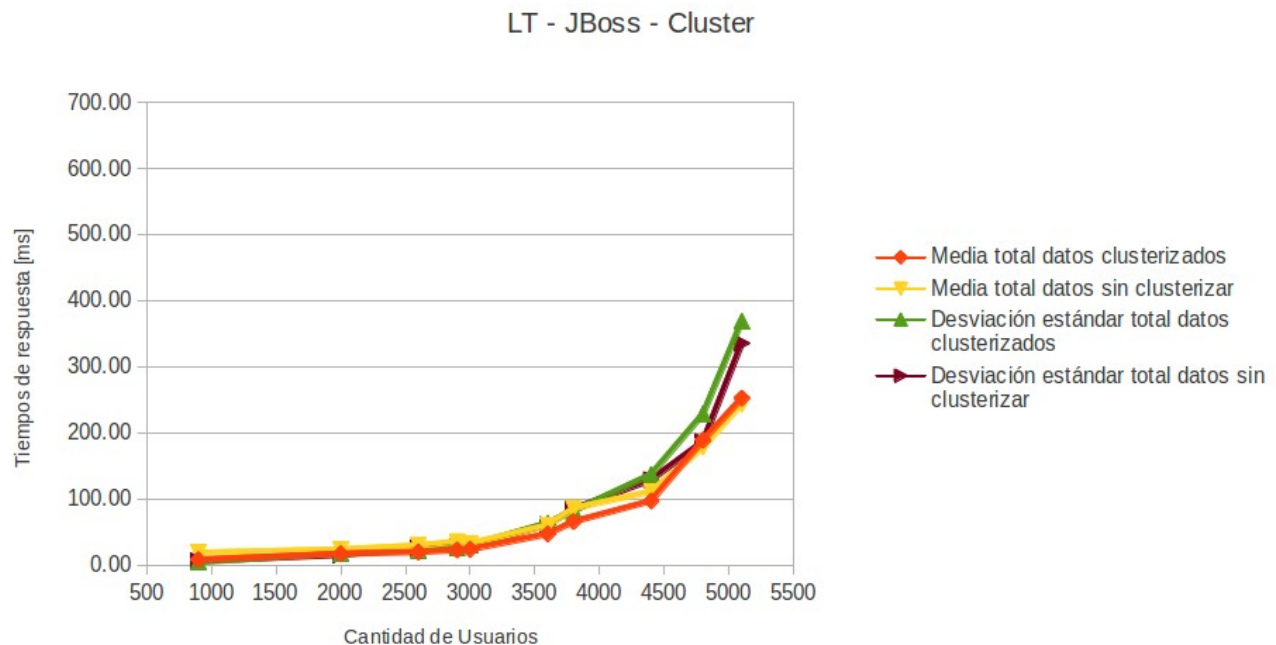


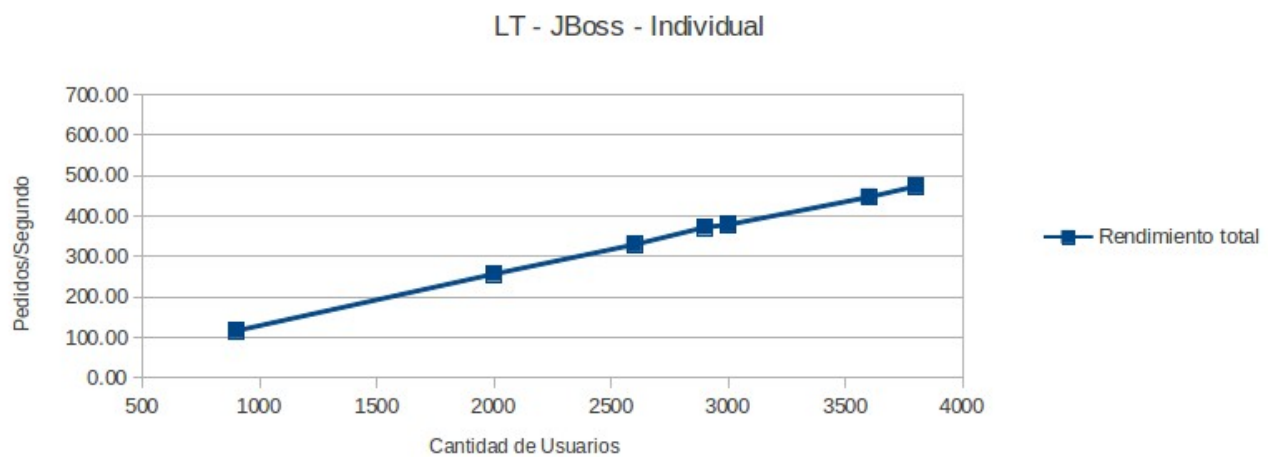
Ilustración 27: LT - JBoss - Cluster

El punto de tiempo estándar más alto en toda la prueba, es de 621,44 milisegundos, para los datos clusterizados con 5100 usuarios, que es un valor aceptable. Se concluye que en toda la prueba, los tiempos fueron los deseables.

Se puede ver que hasta un poco antes de los 4500 usuarios, el comportamiento del Cluster es el deseable, y que incluso aunque lamentablemente el servicio se separe del ideal, de alguna manera llega a atender hasta a 5000 usuarios sin que haya un detrimento en el punto de tiempo estándar; a continuación, el ambiente se queda sin recursos, y colapsa.

Load Test (LT) - Individual

Total hilos	Rendimiento total	Media total datos clusterizados	Media total datos sin clusterizar	Desviación estándar total datos clusterizados	Desviación estándar total datos sin clusterizar
900	116,77	5,33	16,92	4,19	8,24
2000	256,92	4,33	15,79	5,24	8,47
2600	329,77	4,67	17,67	6,81	11,88
2900	372,43	5,00	19,58	8,17	14,72
3000	378,90	7,00	24,50	24,57	32,58
3600	447,12	10,00	32,50	45,42	52,78
3800	473,64	11,33	40,71	55,41	64,83

*Ilustración 28: LT - JBoss - Individual*

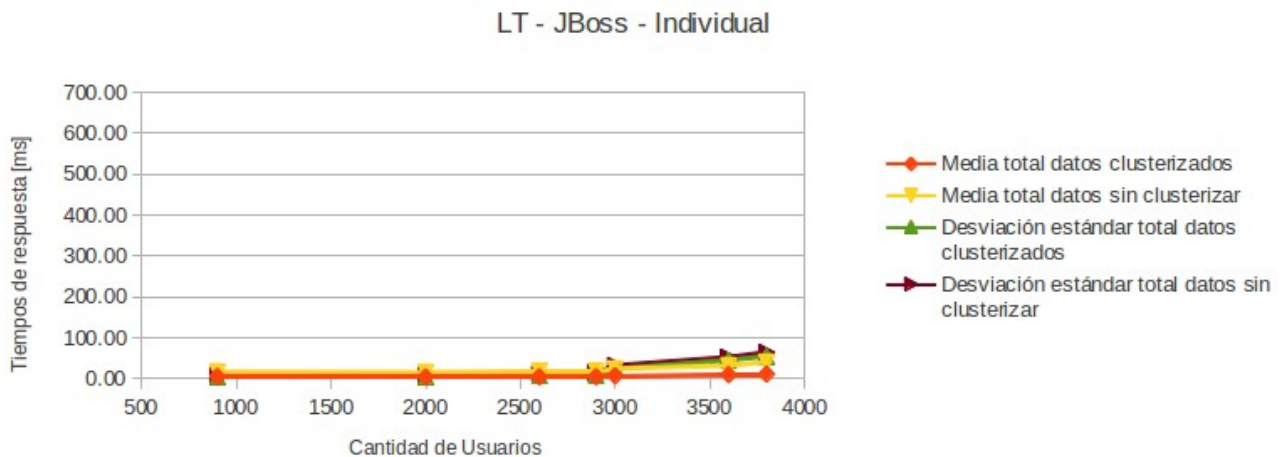


Ilustración 29: LT - JBoss - Individual

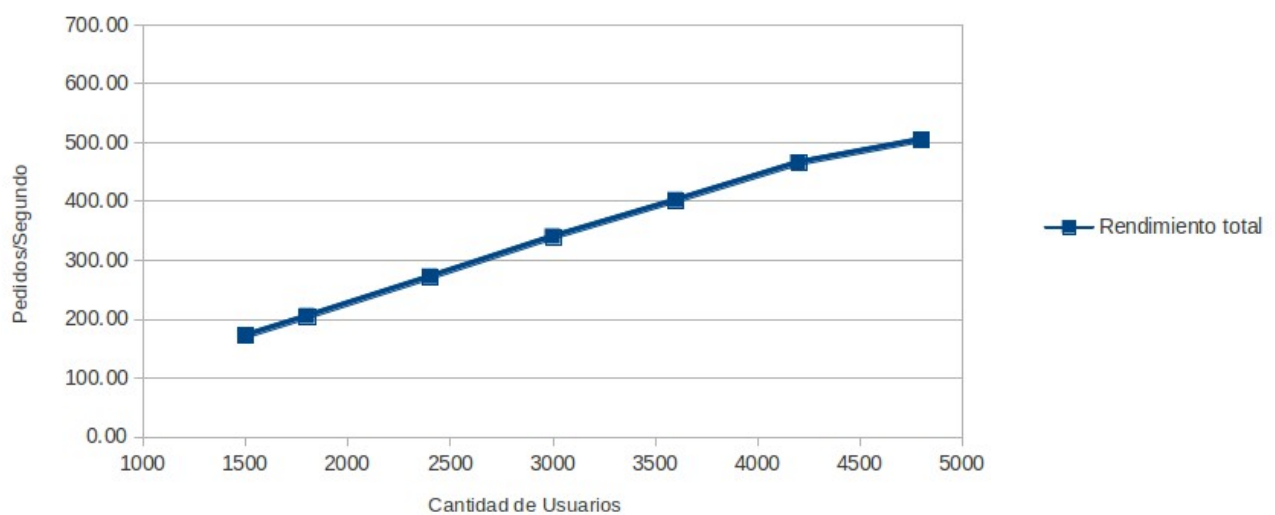
El punto de tiempo estándar más alto en toda la prueba, es de 105,54 milisegundos, para los datos sin clusterizar con 3800 usuarios, que es un valor aceptable. Se concluye que en toda la prueba, los tiempos fueron los deseables.

Mismo patrón de comportamiento que en el caso anterior.

Load Test Datos Clusterizados (ST) - Cluster

Total hilos	Rendimiento total	Media total datos clusterizados	Desviación estándar total datos clusterizados
1500	172,12	10,00	9,77
1800	204,89	10,67	11,26
2400	271,97	12,67	13,94
3000	340,10	16,33	17,61
3600	401,90	39,67	51,51
4200	465,81	61,33	149,78
4800	504,77	428,33	633,86

ST - JBoss - Cluster

*Ilustración 30: ST - JBoss - Cluster*

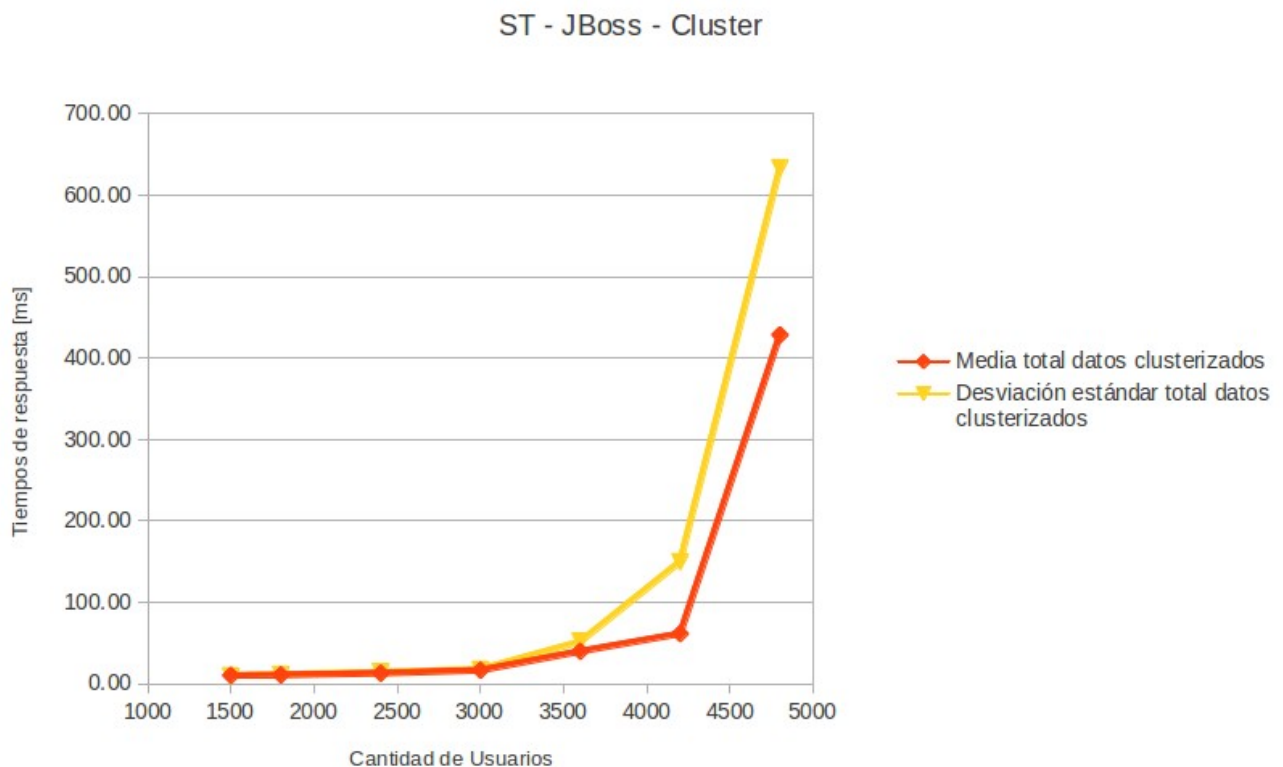


Ilustración 31: ST - JBoss - Cluster

El punto de tiempo estándar con 4200 usuarios, el más alto hasta ese punto, es de 211,11 milisegundos naturalmente para los datos clusterizados que son los únicos datos analizados en esta prueba, y el mismo valor en el paso siguiente de 4800 usuarios salta hasta 1062,19 milisegundos, lo cual excede los tiempos estándar esperados en una aplicación web. De hecho, valores al margen, queda más que claro contemplando el gráfico, que entre 4000 y 4500 usuarios el sistema empieza a colapsar, pero soporta hasta 4800 usuarios. A continuación, el ambiente se queda sin recursos, y colapsa por completo.

Muy similar cualitativamente al caso de Load Test para Cluster de JBoss.

Load Test Datos Clusterizados (ST) - Individual

Total hilos	Rendimiento total	Media total datos clusterizados	Desviación estándar total datos clusterizados
1500	165,65	235,67	276,26
1800	200,27	273,67	688,57
2400	189,42	4.144,33	2.862,55
3000	194,07	6.809,33	4.672,45
3600	195,23	9.418,33	6.259,89
4200	198,77	12.266,00	8.037,87

ST - JBoss - Individual

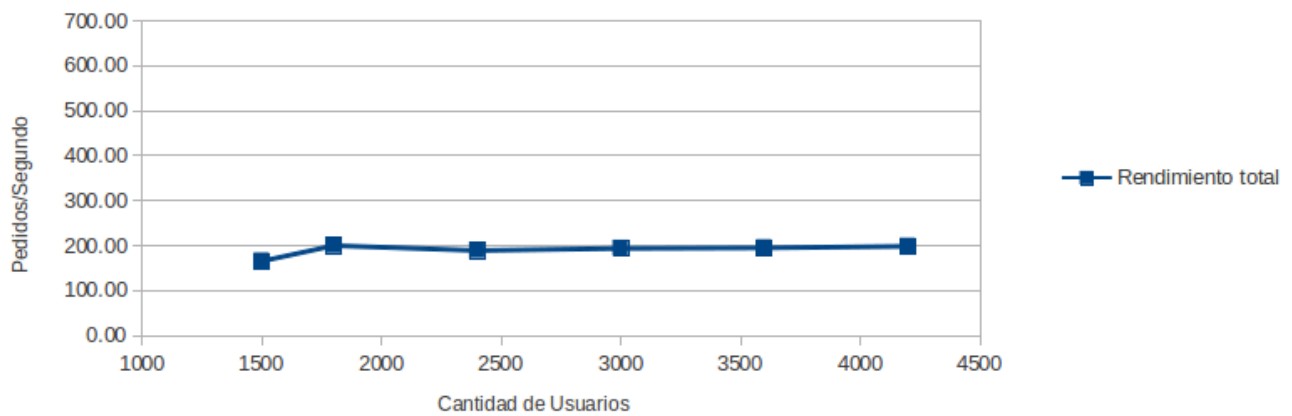


Ilustración 32: ST - JBoss - Individual

ST - JBoss - Individual

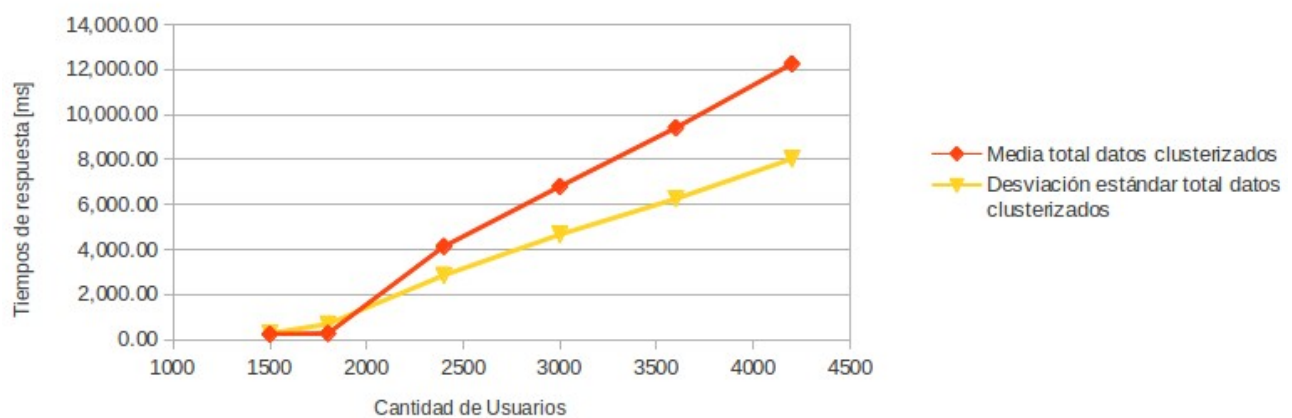


Ilustración 33: ST - JBoss - Individual

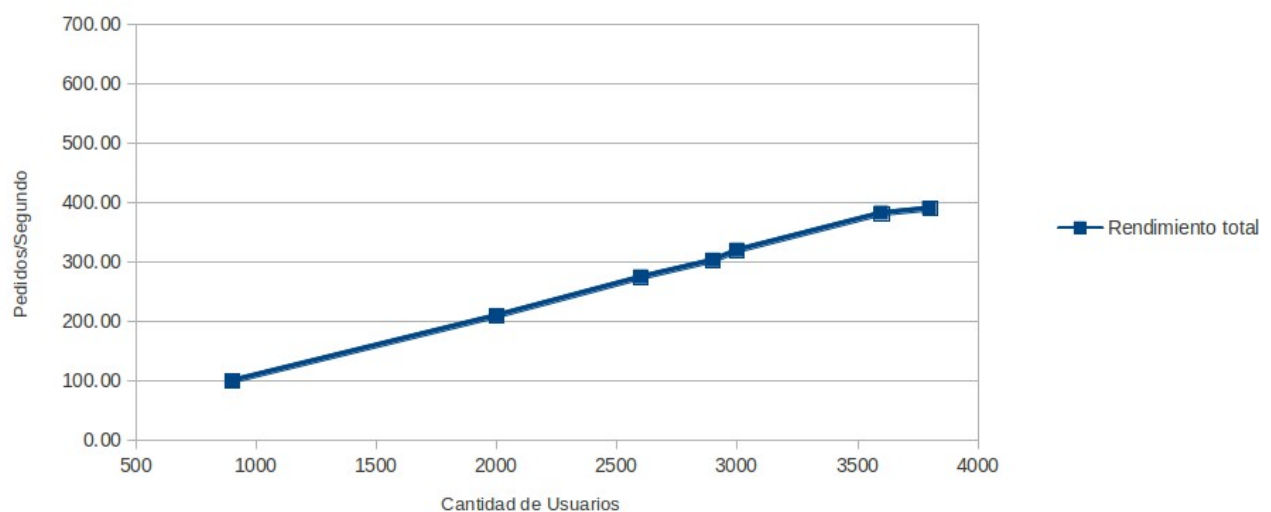
El punto de tiempo estándar con 1800 usuarios, el más alto hasta ese punto, es de 962,24 milisegundos naturalmente para los datos clusterizados que son los únicos datos analizados en esta prueba, lo cual se acerca mucho al límite considerado, y el mismo valor en el paso siguiente de 2400 usuarios salta hasta 7 segundos, lo cual como se está trabajando con promedios indica que se exceden los tiempos estándar esperados en una aplicación web. El punto de tiempo estándar termina alcanzando los 20 segundos a los 4200 usuarios, lo cual es desde el punto de vista de los valores esperados, totalmente inadmisibles. Valores al margen, queda más que claro contemplando el gráfico, que si bien evidentemente los números en el inicio de la prueba ya son elevados en comparación con otras pruebas, entre 1500 y 2000 usuarios el sistema empieza a colapsar, pero soporta hasta 4200 usuarios. A continuación, el ambiente se queda sin recursos, y colapsa por completo.

Sencillamente, esta prueba es excesivamente exigente para un solo nodo, independientemente de que en ese único nodo se encuentre un servidor EE como JBoss. Un ejemplo muy claro del funcionamiento superador de un Cluster, en este caso.

Load Test Datos No Clusterizados (NST) - Cluster

Total hilos	Rendimiento total	Media total datos sin clusterizar	Desviación estándar total datos sin clusterizar
900	99,81	15,67	19,26
2000	209,65	21,33	26,75
2600	274,53	22,00	27,66
2900	302,90	26,67	34,90
3000	319,75	29,33	39,62
3600	382,22	63,33	126,20
3800	390,22	112,67	292,23

NST - JBoss - Cluster

*Ilustración 34: NST - JBoss - Cluster*

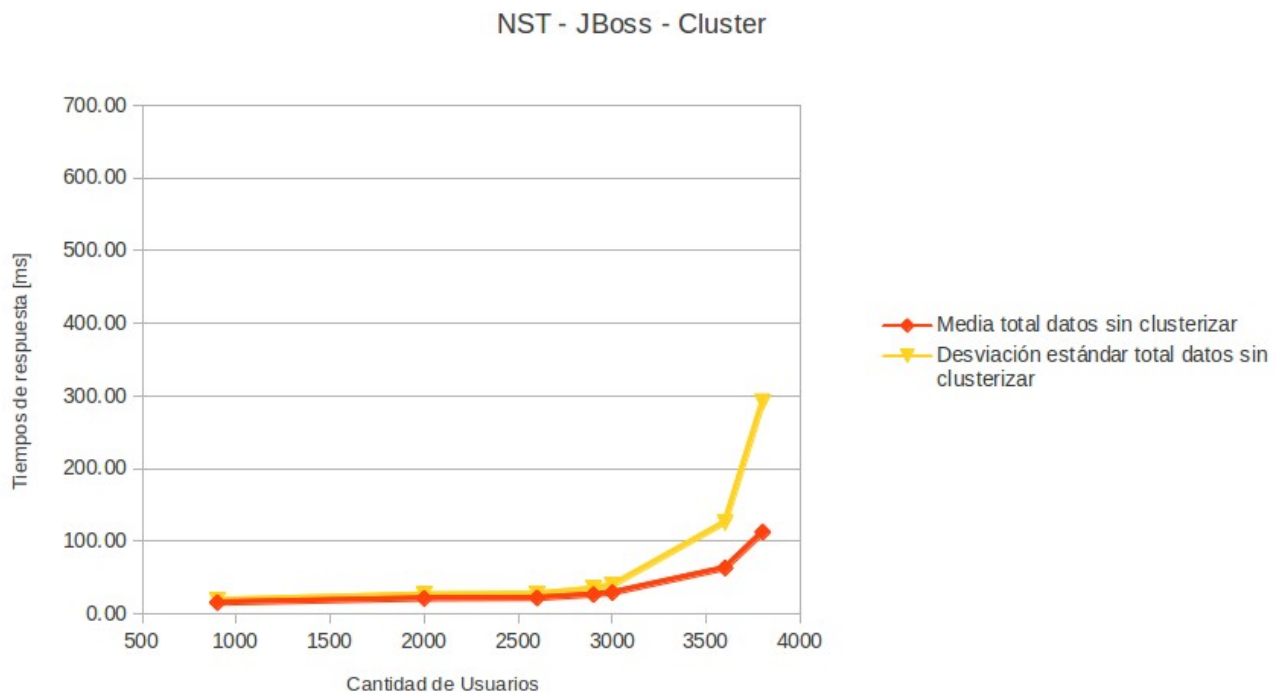


Ilustración 35: NST - JBoss - Cluster

El punto de tiempo estándar más alto en toda la prueba, es de 404,90 milisegundos, para los datos sin clusterizar con 3800 usuarios, que es un valor aceptable. Se concluye que en toda la prueba, los tiempos fueron los deseables.

Hasta aproximadamente los 3500 usuarios, el comportamiento del sistema es prácticamente el ideal; y se ve con claridad que ese es el pico en esta configuración y en esta prueba; no se llega al siguiente paso, el de los 4000 usuarios.

Load Test Datos No Clusterizados (NST) - Individual

Total hilos	Rendimiento total	Media total datos sin clusterizar	Desviación estándar total datos sin clusterizar
900	95,58	12,33	17,14
2000	209,88	85,00	503,74

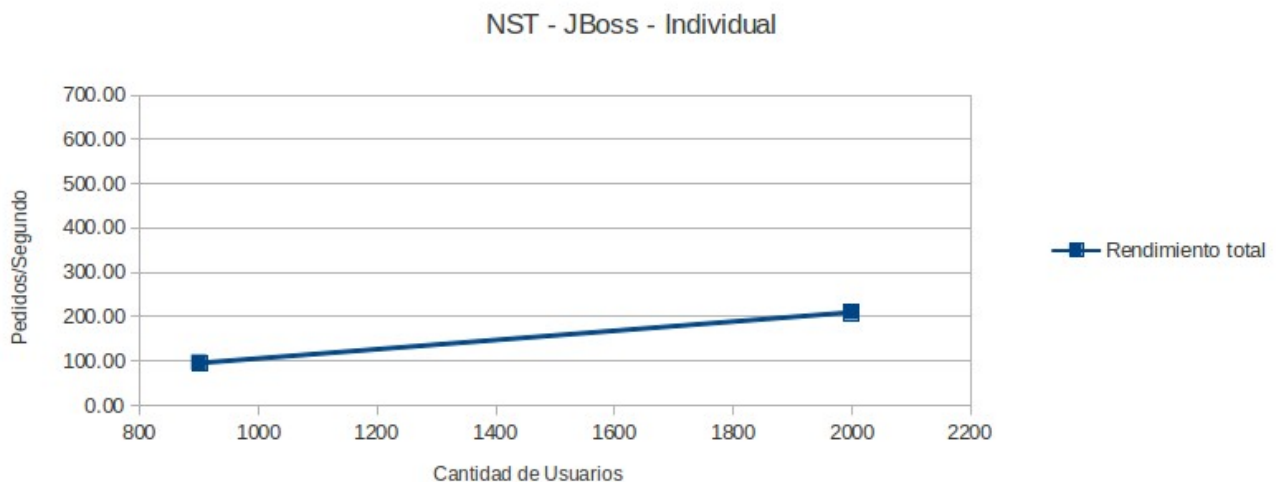


Ilustración 36: NST - JBoss - Individual

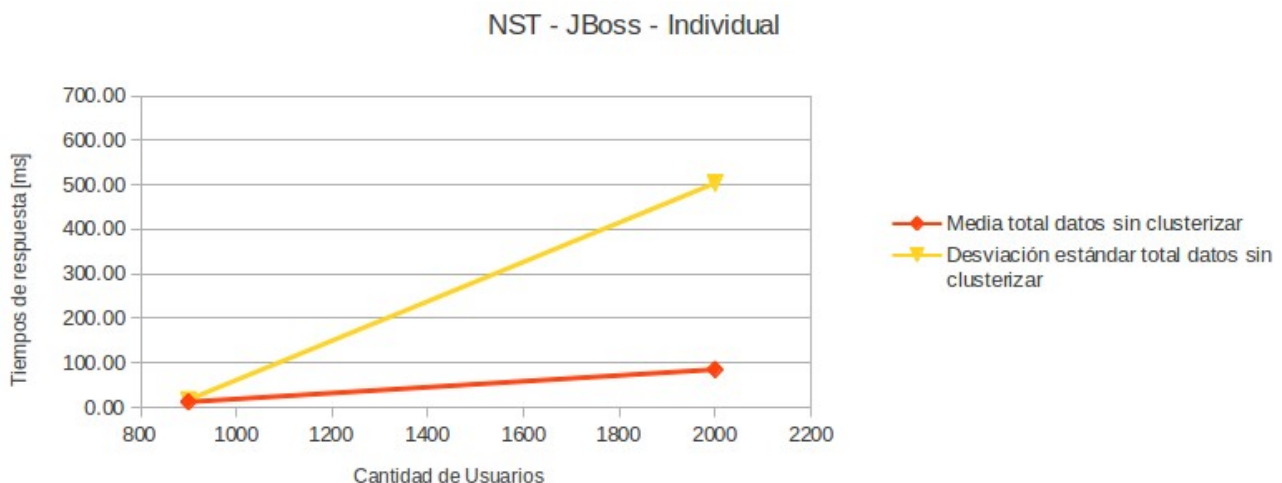


Ilustración 37: NST - JBoss - Individual

El punto de tiempo estándar más alto en toda la prueba, es de 588,74 milisegundos, para los datos sin clusterizar con 2000 usuarios, que es un valor aceptable, independientemente de que la pendiente indica que si se pudieran sumar más pasos el sistema colapsará. Se concluye que en toda la prueba, los tiempos fueron los deseables.

Tal como en el caso de Sync Test, el nodo individual de JBoss no puede tolerar esta prueba.

5.7.2. Terracotta + Tomcat

Pruebas manuales

Siguiendo el mismo razonamiento que en la sección correspondiente de JBoss, planteamos exactamente el mismo caso de prueba con el objetivo de verificar que con este Cluster también se superan las pruebas manuales. Una diferencia con el caso de JBoss, es que en el caso de Terracotta, la información de log se obtiene consultando la consola administrativa de Terracotta a diferencia de JBoss donde necesariamente debemos consultar los archivos de log.

Para más información ver *"7.6. Apéndice 6: Ejecución de la combinación de casos de pruebas manuales"*.

Pruebas automáticas

Stress Test (STT) - Cluster

Total hilos	Rendimiento total	Media total datos clusterizados	Media total datos sin clusterizar	Desviación estándar total datos clusterizados	Desviación estándar total datos sin clusterizar
300	40,56	5,00	16,94	2,86	6,75
600	80,45	4,67	18,58	3,70	7,90
900	118,47	6,00	23,19	8,31	13,18
1100	144,40	8,67	36,02	13,49	39,93
1200	155,26	27,33	107,65	69,40	184,41

STT - Terracotta - Cluster

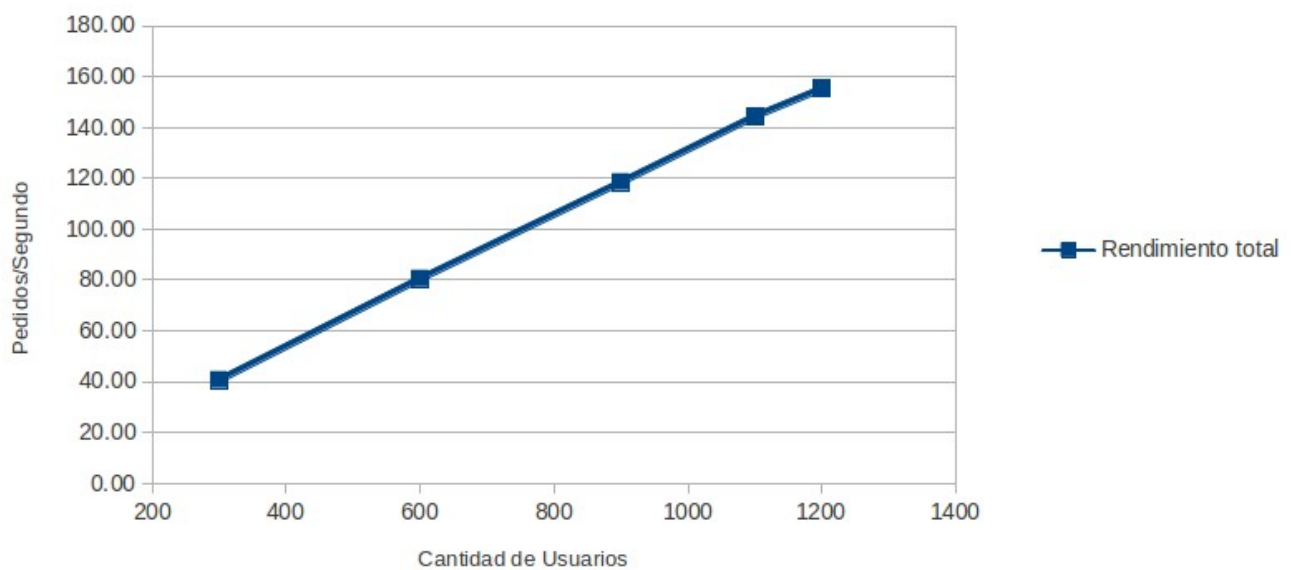


Ilustración 38: STT - Terracotta - Cluster

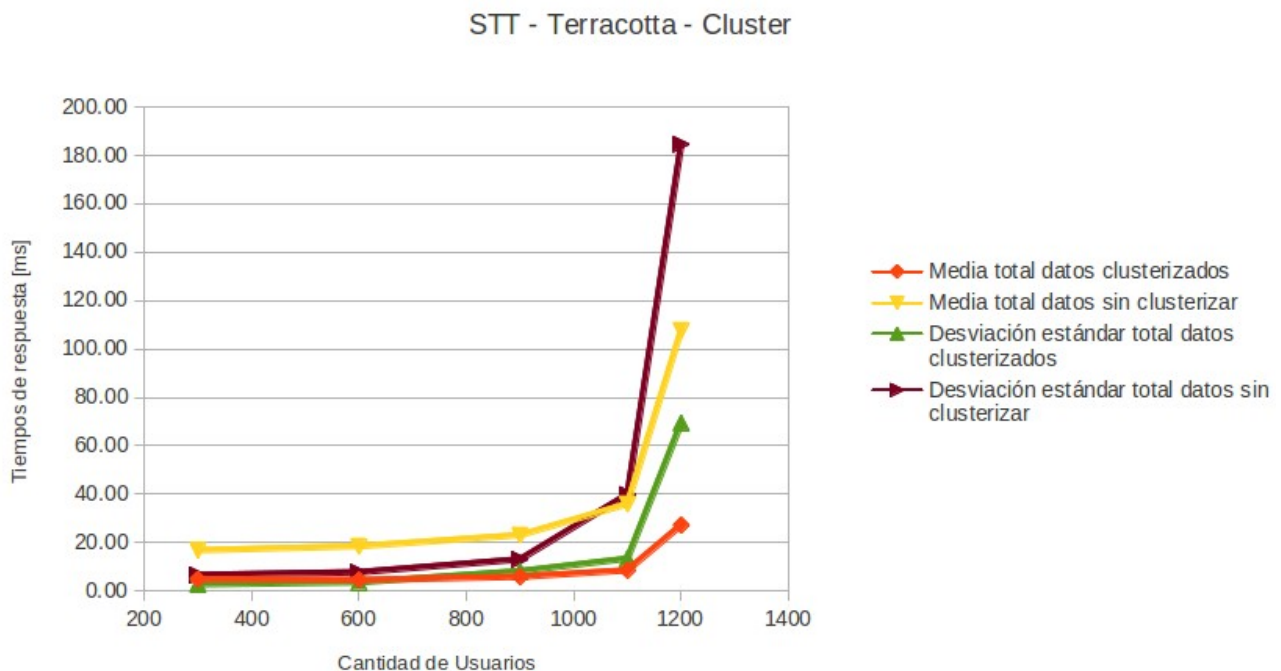


Ilustración 39: STT - Terracotta - Cluster

El punto de tiempo estándar más alto en toda la prueba, es de 292,06 milisegundos, para los datos sin clusterizar con 1200 usuarios, que es un valor aceptable. Se concluye que en toda la prueba, los tiempos fueron los deseables.

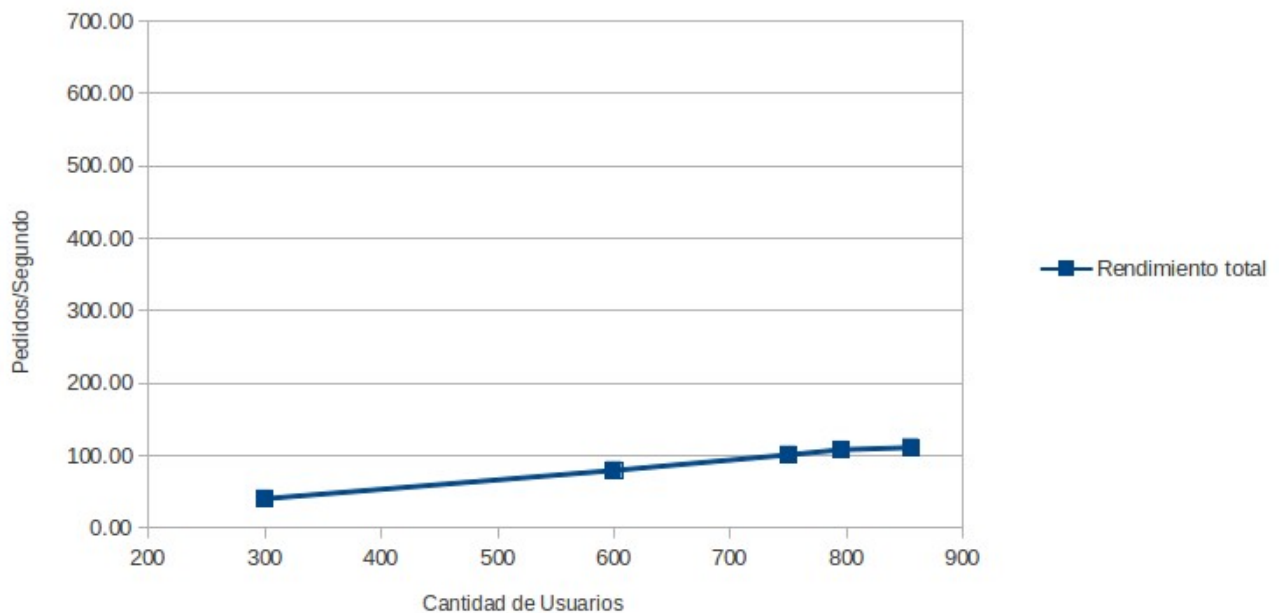
Se puede observar en este caso, un patrón de respuesta muy similar a la prueba STT - JBoss - Cluster, aunque el ambiente colapsa en una cantidad de usuarios menor que el Cluster de JBoss; además, se vé una característica que se notará en la mayoría de las pruebas de Terracotta + Tomcat, que es que mientras Terracotta + Tomcat tiene recursos del nodo en que está trabajando, el tiempo de respuesta es muy pequeño, o sea, es muy bueno; sin embargo, no va empeorando su performance de manera gradual, sino que de un salto pasa a un estado en que ya no puede responder; JBoss como se puede ver en los gráficos, es más gradual.

Creemos haber demostrado que esta característica, se debe a que el Cluster con JBoss demanda muchos más recursos de hardware y de conectividad, por el tipo de tecnología y servicios que ofrece JBoss. Para ejemplificar: no es lo mismo sincronizar con JGroups (JBoss), a simplemente, no sincronizar (ver *Sync Test - Cluster*).

Otra particularidad a tener en cuenta, es que en el caso de JBoss los datos clusterizados fueron siempre los que entregaron el pico en el punto de tiempo estándar, mientras que en este caso de Terracotta + Tomcat son los datos sin clusterizar los que dan ese pico. Concluimos que esto se debe a que con Terracotta se da una combinación muy particular al disponer de una muy buena sincronización, sin demandar los recursos que demanda JBoss, lo cual le permite tener un rendimiento incluso superior a Tomcat en este caso, que ya de por sí es mucho más liviano que JBoss.

Stress Test (STT) - Individual

Total hilos	Rendimiento total	Media total datos clusterizados	Media total datos sin clusterizar	Desviación estándar total datos clusterizados	Desviación estándar total datos sin clusterizar
300	40,24	3,33	15,63	2,79	7,59
600	79,28	3,00	17,79	3,22	11,01
750	100,87	4,33	25,77	13,43	26,14
795	107,91	3,67	25,56	10,02	22,29
855	110,94	13,67	49,06	261,66	184,11

STT - Terracotta - Individual*Ilustración 40: STT - Terracotta - Individual*

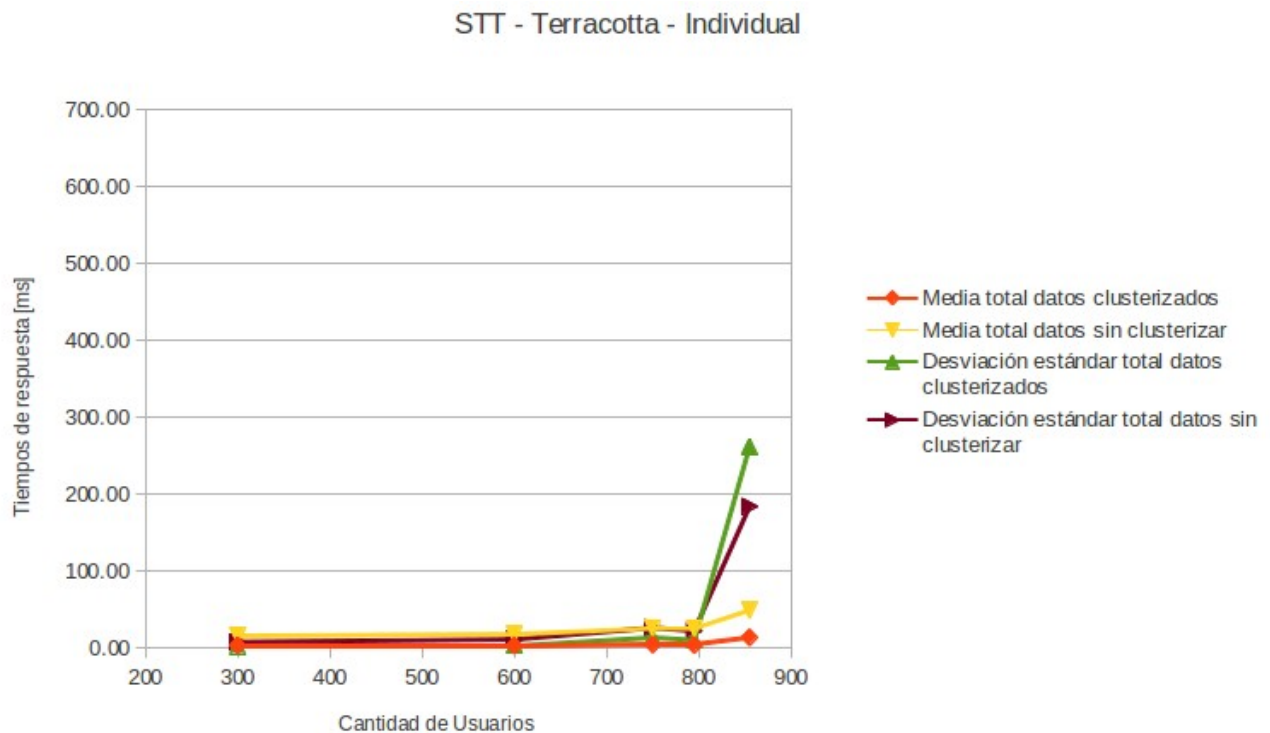


Ilustración 41: STT - Terracotta - Individual

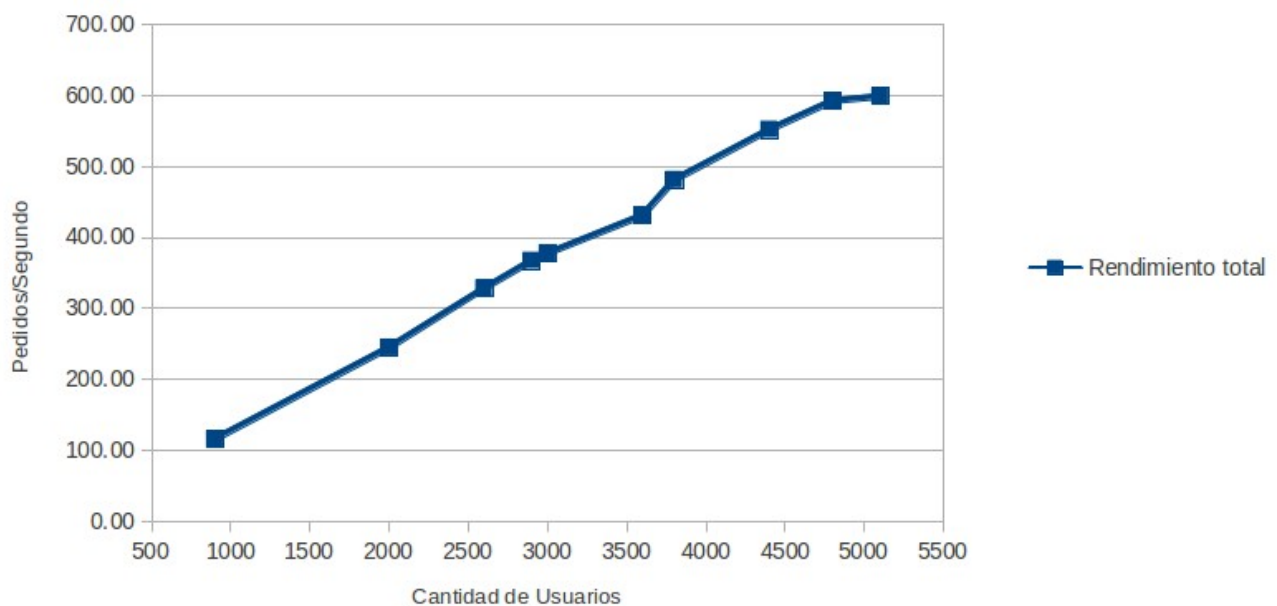
El punto de tiempo estándar más alto en toda la prueba, es de 275,32 milisegundos, para los datos clusterizados con 855 usuarios, que es un valor aceptable. Se concluye que en toda la prueba, los tiempos fueron los deseables.

Se repite el patrón visto anteriormente: mientras el Cluster Terracotta + Tomcat puede responder, su tiempo de respuesta es excelente, y de repente colapsa.

Se percibe también que en este caso, no se llegan a atender a tantos usuarios como en la prueba equivalente en el JBoss individual.

Load Test (LT) - Cluster

Total hilos	Rendimiento total	Media total datos clusterizados	Media total datos sin clusterizar	Desviación estándar total datos clusterizados	Desviación estándar total datos sin clusterizar
900	116,14	4,00	16,29	3,77	24,77
2000	245,14	5,67	21,00	9,60	28,86
2600	328,26	5,00	18,58	7,05	7,82
2900	367,43	6,00	22,00	8,35	12,66
3000	377,58	6,00	20,96	8,27	10,72
3600	431,32	9,00	33,92	20,67	30,99
3800	481,33	9,00	29,75	16,39	23,09
4400	552,01	14,67	43,25	31,08	40,81
4800	592,65	26,67	76,88	57,02	111,89
5100	598,80	88,67	268,21	1.092,89	1.644,81

LT - Terracotta - Cluster*Ilustración 42: LT - Terracotta - Cluster*

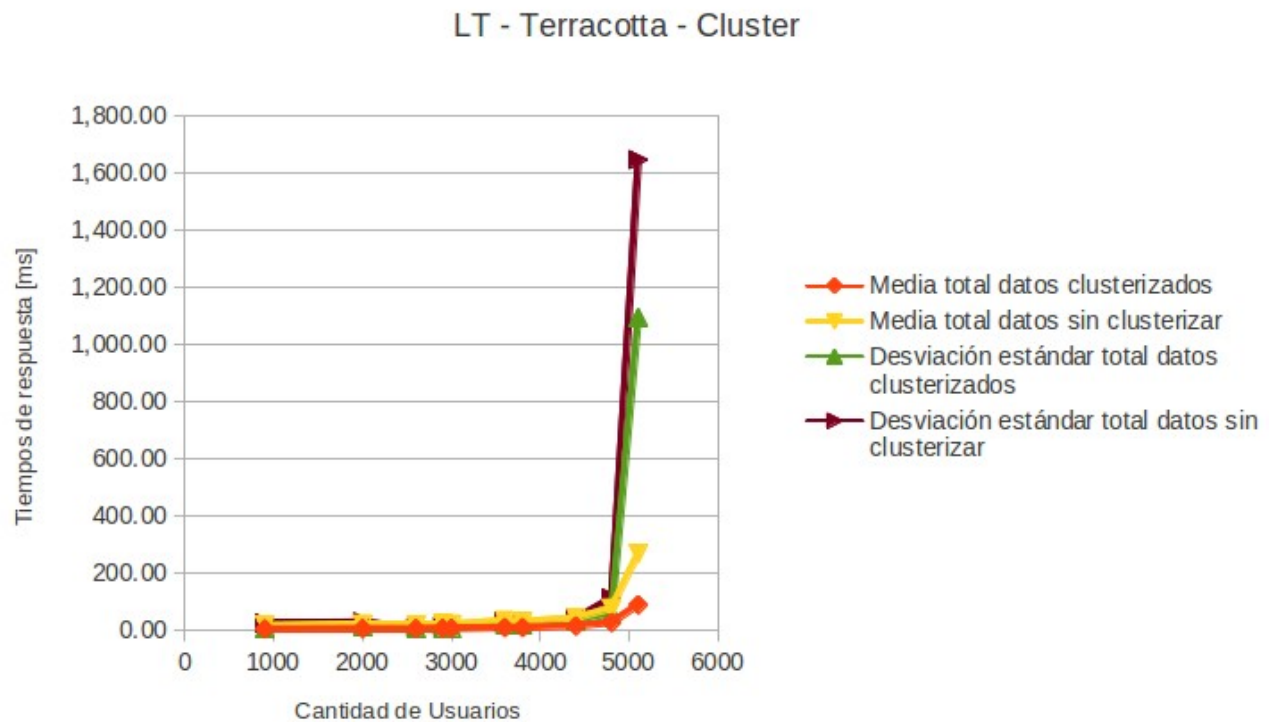


Ilustración 43: LT - Terracotta - Cluster

El punto de tiempo estándar con 4800 usuarios, el más alto hasta ese punto, es de 188,77 milisegundos para los datos sin clusterizar, y el mismo valor en el paso siguiente de 5100 usuarios salta hasta 1913,02 milisegundos, lo cual excede los tiempos estándar esperados en una aplicación web. De hecho, valores al margen, queda más que claro contemplando el gráfico, que entre 4000 y 5000 usuarios el sistema empieza a colapsar, pero soporta hasta 5100 usuarios. A continuación, el ambiente se queda sin recursos, y colapsa por completo.

Se repiten todos los patrones observados.

Load Test (LT) - Individual

Total hilos	Rendimiento total	Media total datos clusterizados	Media total datos sin clusterizar	Desviación estándar total datos clusterizados	Desviación estándar total datos sin clusterizar
900	115,07	2,33	14,75	2,92	8,06
2000	253,32	2,67	20,00	4,24	10,70
2600	330,73	3,67	26,29	5,66	17,23
2900	366,46	5,33	34,17	9,34	29,44
3000	373,67	6,00	35,67	16,81	37,96
3600	436,05	71,00	148,29	1.264,65	1.298,18

LT - Terracotta - Individual

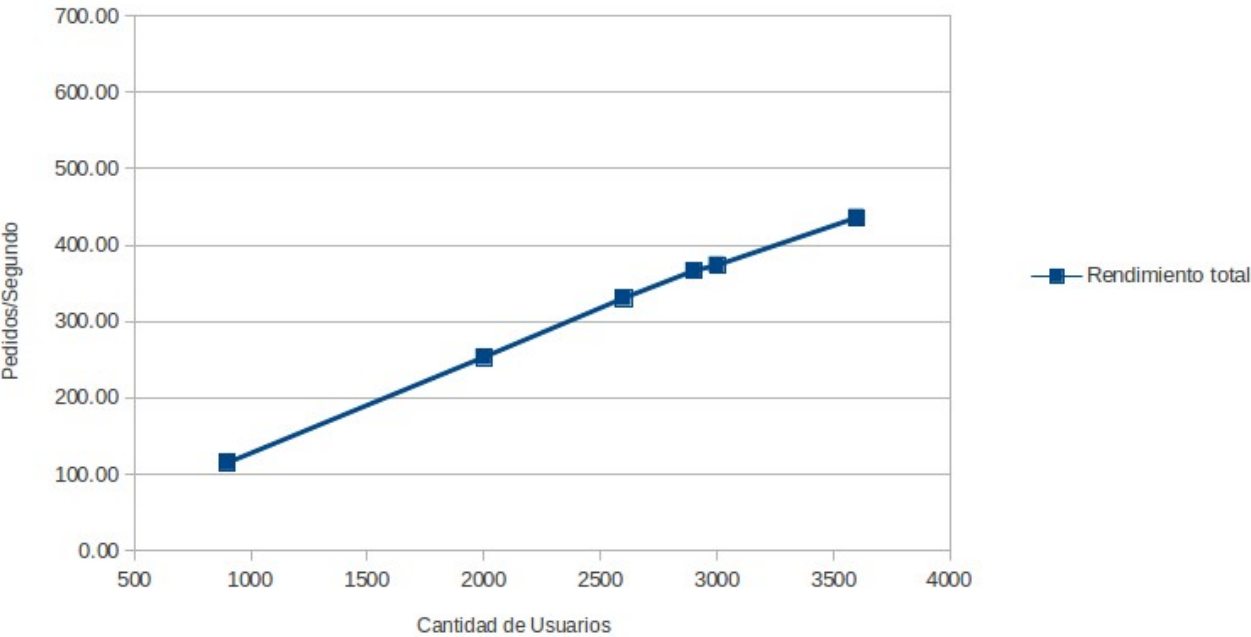


Ilustración 44: LT - Terracotta - Individual

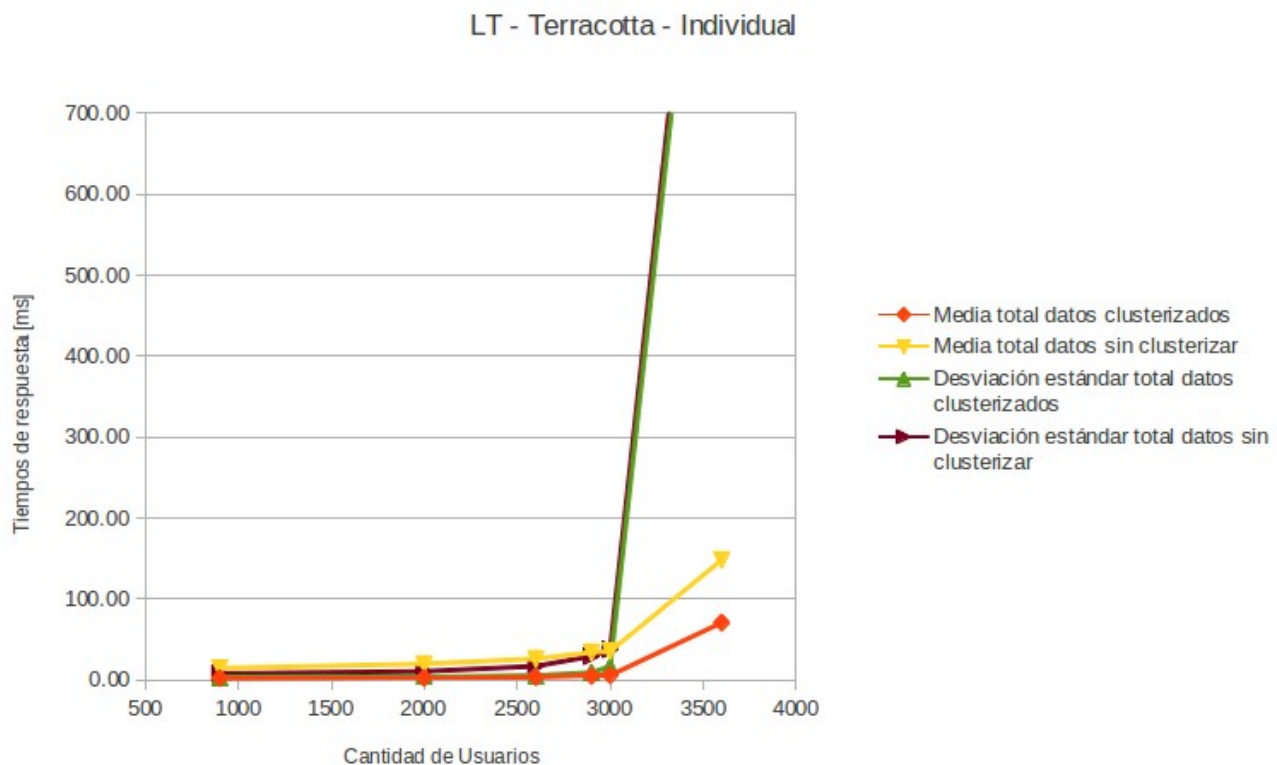


Ilustración 45: LT - Terracotta - Individual

El punto de tiempo estándar con 3000 usuarios, el más alto hasta ese punto, es de 73,63 milisegundos, y el mismo valor en el paso siguiente de 3600 usuarios salta hasta 1446,47 milisegundos, lo cual excede los tiempos estándar esperados en una aplicación web. De hecho, valores al margen, queda más que claro contemplando el gráfico, que a los 3000 usuarios el sistema empieza a colapsar, pero soporta hasta 3600 usuarios. A continuación, el ambiente se queda sin recursos, y colapsa por completo.

Se repiten todos los patrones observados.

Load Test Datos Clusterizados (ST) - Cluster

Total hilos	Rendimiento total	Media total datos clusterizados	Desviación estándar total datos clusterizados
1500	170,82	4,67	7,56
1800	203,81	4,33	6,75
2400	269,83	5,33	9,03
3000	332,43	6,33	10,34
3600	406,23	9,33	17,69
4200	469,55	13,00	23,20
4800	537,05	18,33	38,05
5400	600,40	27,33	55,91
6000	649,73	51,67	114,61

ST - Terracotta - Cluster

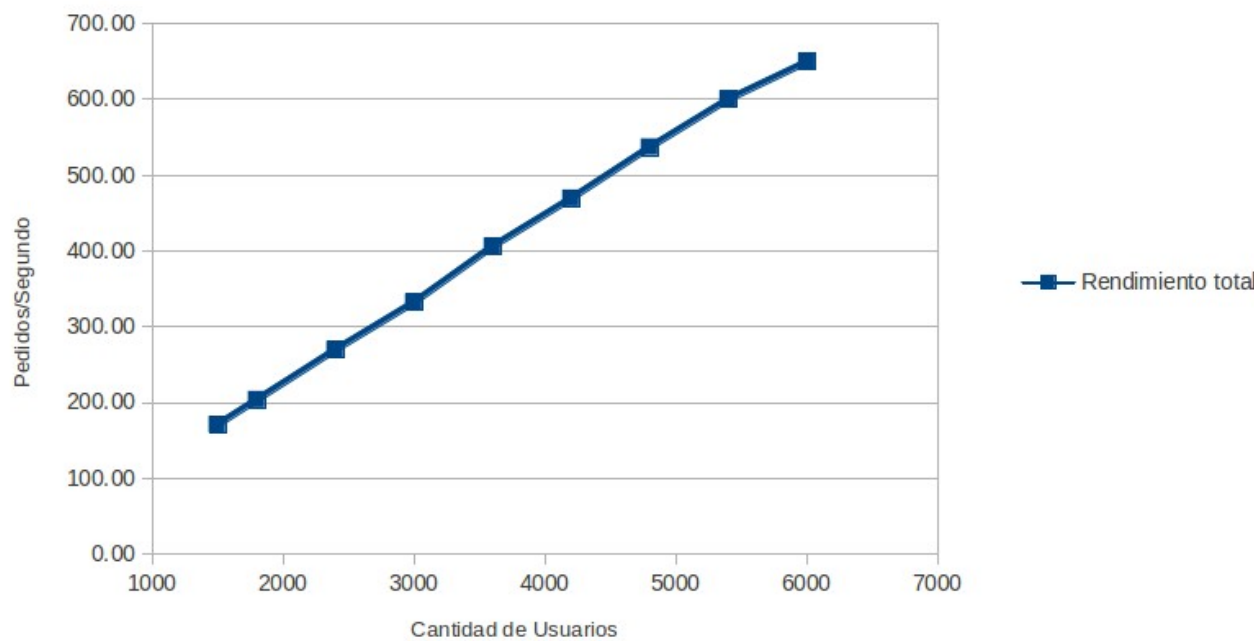


Ilustración 46: ST - Terracotta - Cluster

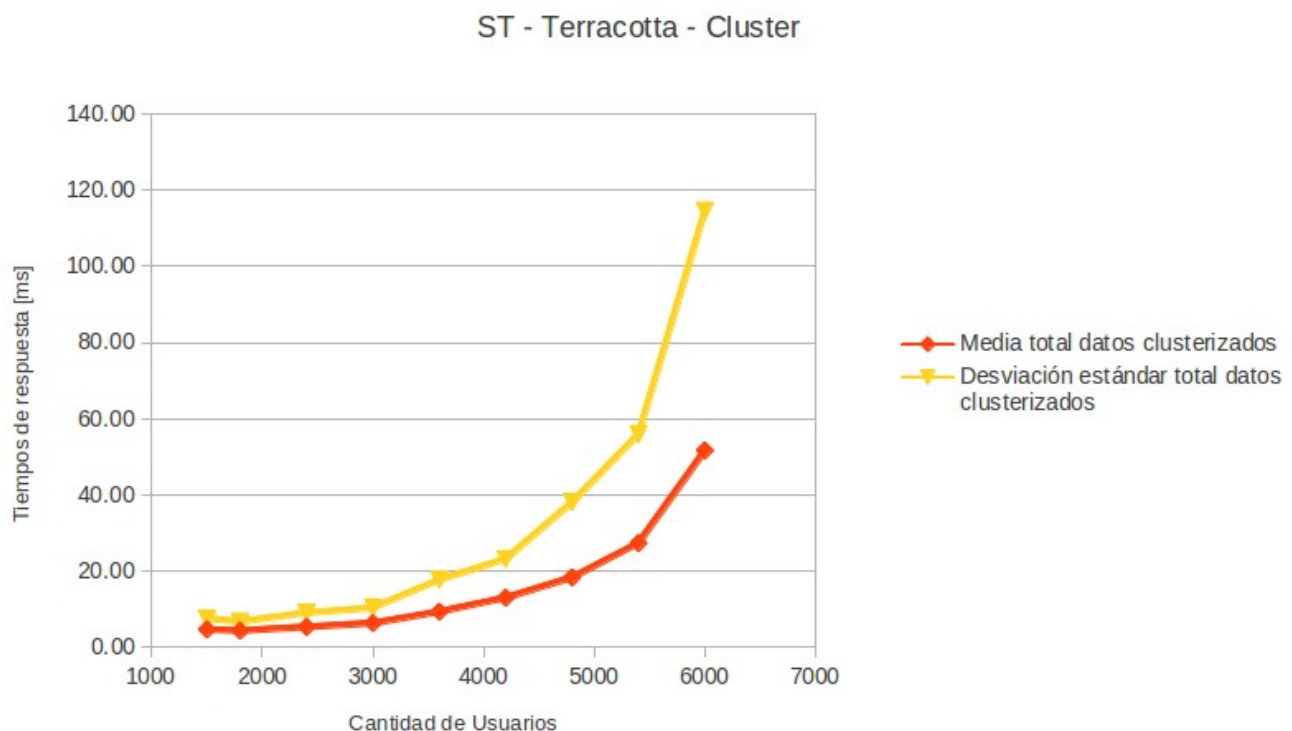


Ilustración 47: ST - Terracotta - Cluster

El punto de tiempo estándar más alto en toda la prueba, es de 166,28 milisegundos, para los datos clusterizados con 6000 usuarios, que es un valor aceptable. Se concluye que en toda la prueba, los tiempos fueron los deseables.

Sin dudas, este es el caso más especial y particular de todos. Es el único caso en que el desempeño global del Cluster con Terracotta + Tomcat, es notablemente superior al desempeño del Cluster con JBoss y la misma prueba.

Tal como se comenta anteriormente, creemos haber demostrado sobre todo con esta gráfica, que esto se debe a que el Cluster con Terracotta - Tomcat no requiere tantos recursos para las comunicaciones como el Cluster con JBoss.

Load Test Datos Clusterizados (ST) - Individual

Total hilos	Rendimiento total	Media total datos clusterizados	Desviación estándar total datos clusterizados
1500	174,10	2,33	2,70
1800	201,74	2,33	4,54
2400	271,24	4,67	40,83
3000	338,27	3,33	6,30
3600	412,26	4,00	5,29
4200	471,24	5,33	8,78
4800	538,61	6,33	10,69
5400	602,89	8,67	14,92
6000	667,29	12,00	23,91
6600	725,20	20,33	82,02
7350	782,16	44,33	628,65

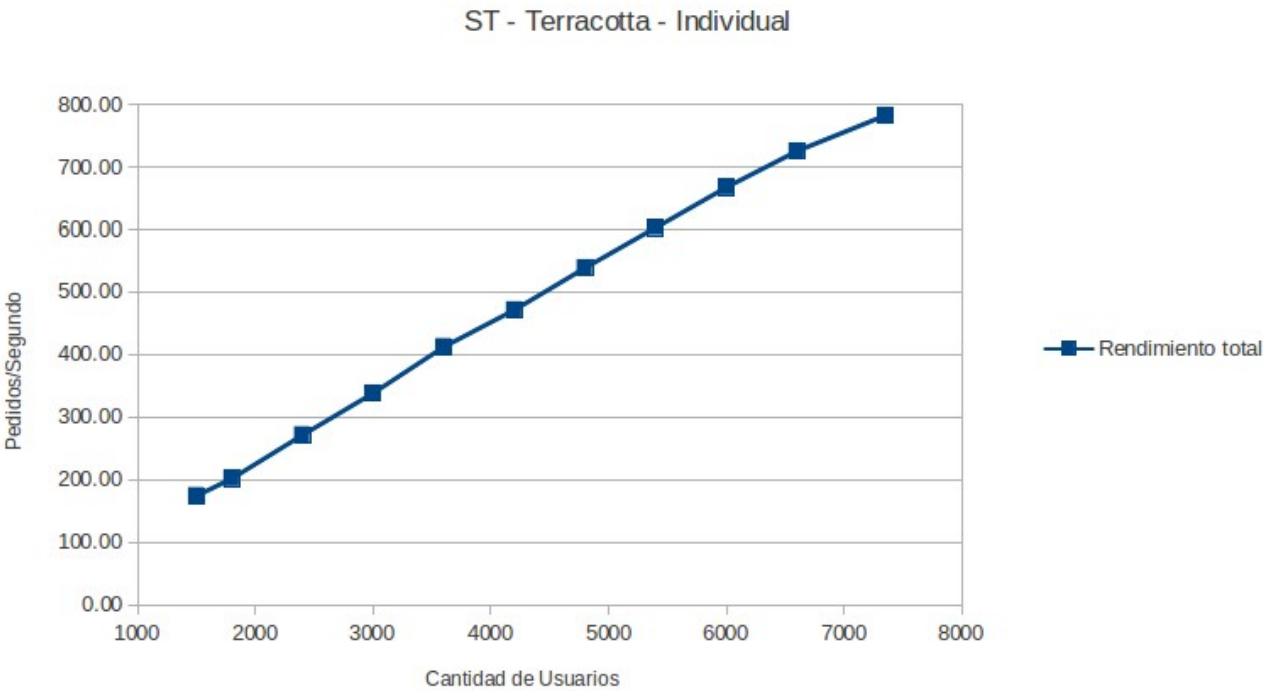


Ilustración 48: ST - Terracotta - Individual

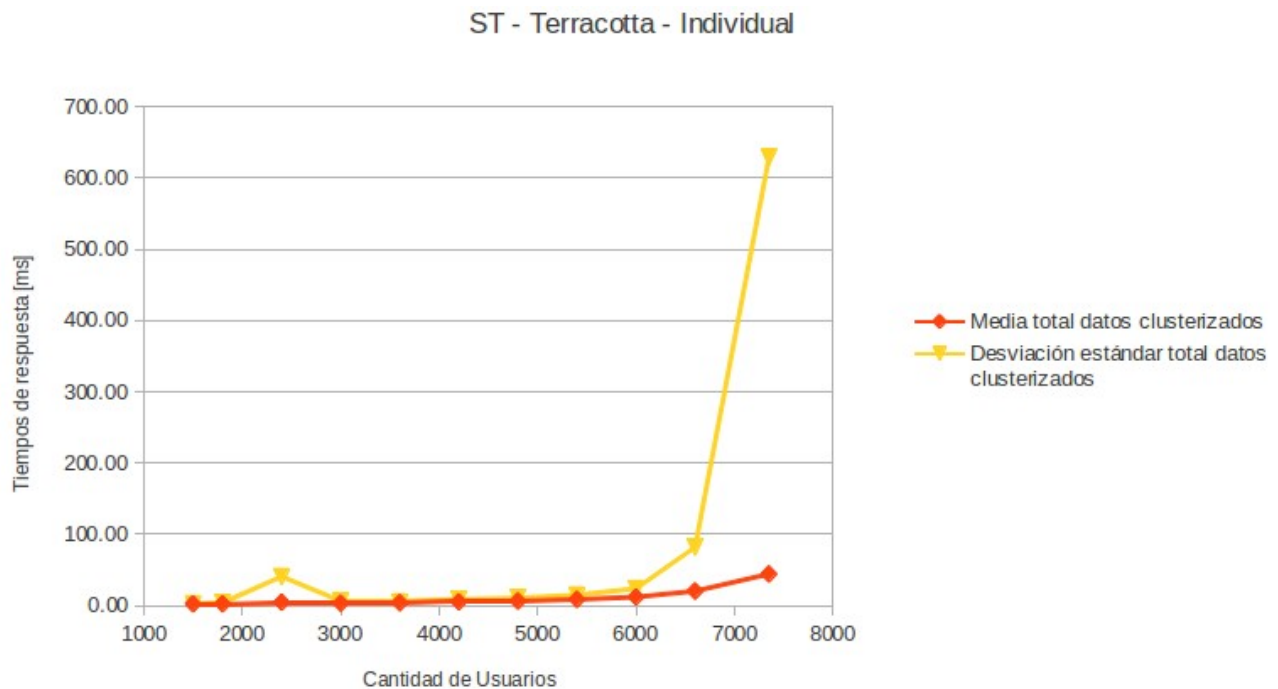


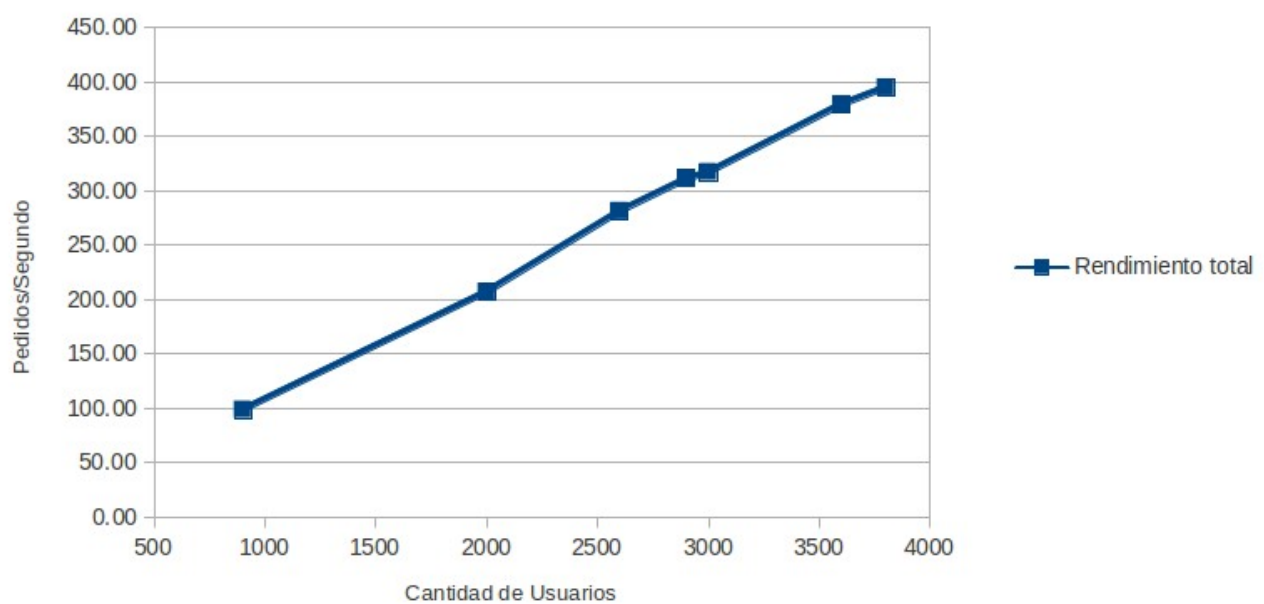
Ilustración 49: ST - Terracotta - Individual

El punto de tiempo estándar con 6600 usuarios, el más alto hasta ese punto, es de 102,35 milisegundos, y el mismo valor en el paso siguiente de 7350 usuarios salta hasta 672,98 milisegundos; si bien el salto es considerable, se sigue estando dentro de los tiempos estándar.

Se repite el caso anterior, en que el Cluster con Terracotta - Tomcat exhibe unos números notables, llegando a atender a más de 7000 usuarios.

Load Test Datos No Clusterizados (NST) - Cluster

Total hilos	Rendimiento total	Media total datos sin clusterizar	Desviación estándar total datos sin clusterizar
900	98,97	17,67	27,18
2000	207,44	20,33	24,44
2600	281,65	23,67	28,65
2900	311,49	27,67	36,57
3000	317,40	30,00	41,08
3600	379,87	166,00	629,73
3800	395,35	310,00	1.049,89

NST - Terracotta - Cluster*Ilustración 50: NST - Terracotta - Cluster*

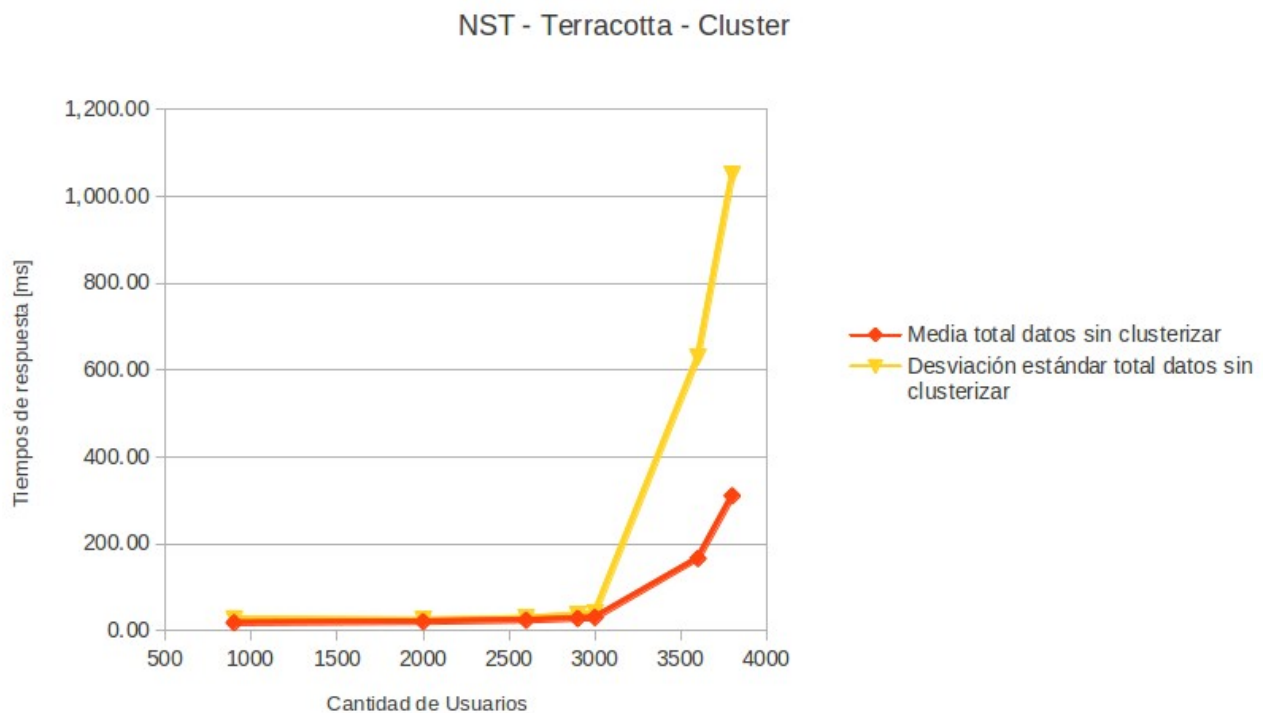


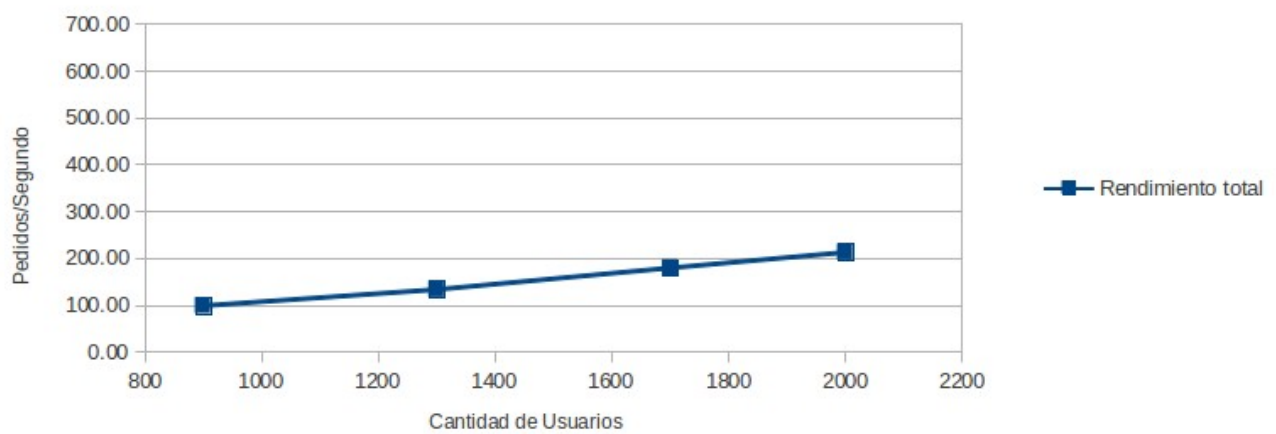
Ilustración 51: NST - Terracotta - Cluster

El punto de tiempo estándar con 3600 usuarios, el más alto hasta ese punto, es de 795,73 milisegundos, y el mismo valor en el paso siguiente de 4800 usuarios salta hasta 1359,89 milisegundos, lo cual excede los tiempos estándar esperados en una aplicación web. De hecho, valores al margen, queda más que claro contemplando el gráfico, que a los 3000 usuarios el sistema empieza a colapsar, pero soporta hasta 3800 usuarios. A continuación, el ambiente se queda sin recursos, y colapsa por completo.

Se retorna a los patrones de comportamiento más frecuentes.

Load Test Datos No Clusterizados (NST) - Individual

Total hilos	Rendimiento total	Media total datos sin clusterizar	Desviación estándar total datos sin clusterizar
900	99,25	14,67	17,35
1300	134,05	18,00	21,92
1700	180,15	27,00	47,27
2000	213,34	132,00	506,79

NST - Terracotta - Individual*Ilustración 52: NST - Terracotta - Individual*

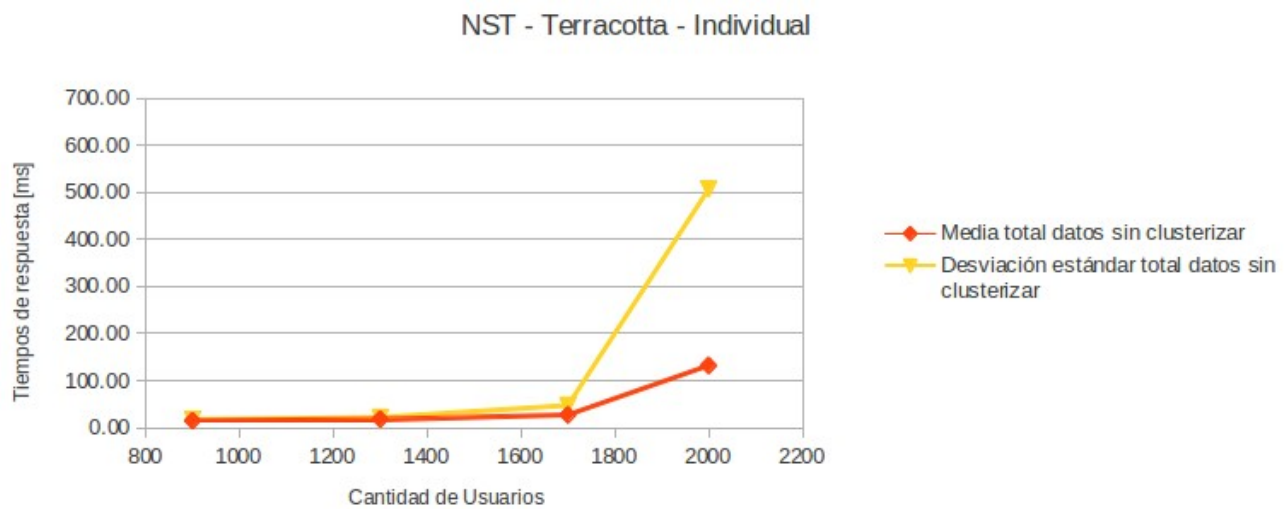


Ilustración 53: NST - Terracotta - Individual

El punto de tiempo estándar con 1700 usuarios, el más alto hasta ese punto, es de 74,27 milisegundos, y el mismo valor en el paso siguiente de 2000 usuarios salta hasta 638,79 milisegundos; si bien el salto es considerable, se sigue estando dentro de los tiempos estándar de respuesta.

Se retorna a los patrones de comportamiento más frecuentes.

6. Conclusiones

Podemos analizar los resultados a partir de las propiedades de arquitectura perseguidas:

- Buena performance: con ambas herramientas conseguimos igualmente buenos resultados en cuanto a tiempos de respuesta, y rendimiento. Incluso se estuvo en casi todas las pruebas, dentro de los tiempos estándar de respuesta de una aplicación web.

Gracias a los ajustes explicados anteriormente, se ha llegado a acomodar una cantidad máxima de usuarios entre los 4500 y 6000, según la prueba. Si se tiene en cuenta que nunca dispusimos de un servidor real, y que al mismo tiempo los servidores debían correr la aplicación de pruebas (la cual consume muchos recursos), podemos conjeturar que el planteo de la aplicación de pruebas que es una aplicación que aporta servicios en una institución educativa de gran tamaño, y que a modo de comparación se toma el caso de la Facultad de Ingeniería de la Universidad de Buenos Aires con sus 8000 alumnos, creemos que se puede decir que se cumplieron los objetivos de servicio masivo planteados.

- Alta disponibilidad: de las pruebas generales, podemos decir que ante una falla de un nodo del servidor, ambas arquitecturas logran redirigir el tráfico sin problemas, con total transparencia para el usuario. Por otro lado, también podemos relacionar la buena performance obtenida con la alta disponibilidad, ya que al soportar una gran cantidad de usuarios, no se producen cuellos de botella fácilmente.
- Escalabilidad: ambas herramientas probaron proveer de buena escalabilidad, ya que en ambos casos es posible con bastante facilidad agregar nuevos nodos al Cluster, y así soportar diferentes niveles de carga del sistema.

- Elasticidad: en los siguientes gráfico presentamos la comparativa de los resultados de las pruebas de carga para ambas herramientas:

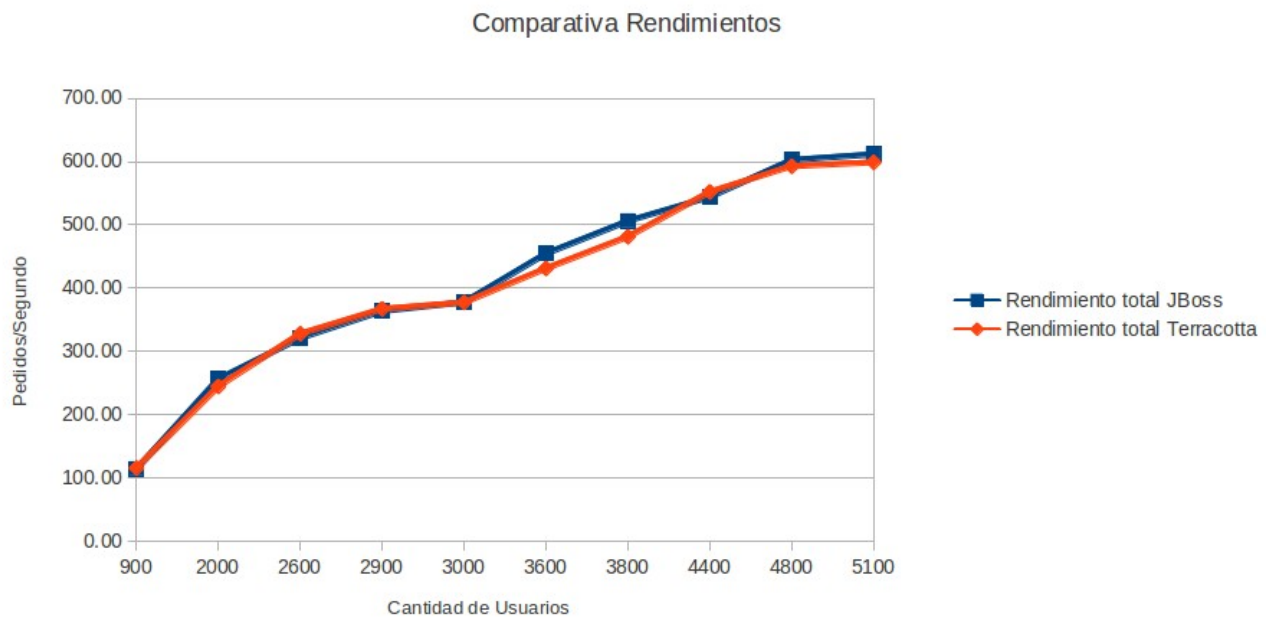


Ilustración 54: Comparativa Rendimientos

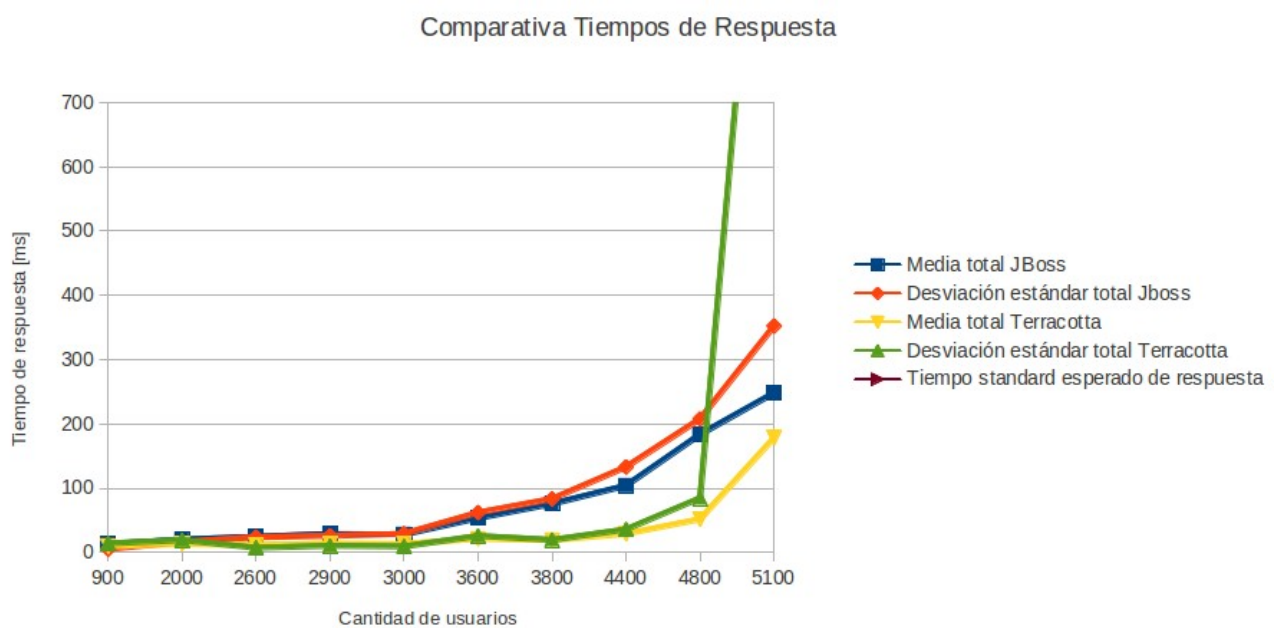


Ilustración 55: Comparativa Tiempos de Respuesta

Observando la performance y redimiento conseguidos con ambas herramientas, podemos ver que JBoss es mas **predecible**, ya que a medida que aumentamos la carga, los tiempos de respuesta se van incrementando paulatinamente. Por el contrario, si bien Terracotta + Tomcat nos provee de una performance similar, al aumentar la carga sobre el sistema llegamos a un punto de quiebre, en donde los tiempos de respuesta se disparan. Teniendo esto en cuenta, podemos decir que la arquitectura implementada con JBoss es una arquitectura mas elástica que la implementada con Terracotta + Tomcat ya que al ser mas predecible se pueden tomar las acciones necesarias para aumentar la cantidad de nodos del Cluster y asi obtener una mayor tolerancia al incremento en la carga.

7. Apéndices

7.1. Apéndice 1: Cómo utilizar las herramientas seleccionadas

General

Corresponde mencionar que durante el desarrollo del presente trabajo, se utilizó una serie de IPs normalizados de cada uno de los componentes y nodos necesarios, declarada en el archivo `/etc/hosts`, a saber:

- terracotta
- basededatos
- apache
- worker1
- worker2

Por ejemplo, en una corrida puntual:

```
192.168.1.101 terracotta
192.168.1.100 basededatos
192.168.1.101 apache
192.168.1.101 worker1
192.168.1.107 worker2
```

Según las tareas a ejecutar o las computadoras e infraestructura de red disponibles, los citados alias tomaron el valor de IP correspondiente a lo que nuestro escenario determinó o necesitó.

Esto nos permitió trabajar los aspectos de configuración de todo el trabajo según dichos alias, evitando tener que mantener configuraciones dedicadas a tal o cual IP "numérico" en particular o a tal o cual computadora en especial.

Como se explicó en las secciones pertinentes, el escenario de pruebas de Cluster siempre estuvo conformado por dos nodos, independientemente de la implementación en sí del Cluster (JBoss - Terracotta / Tomcat). Por ende, se consideró disponer de dos instalaciones de cada servidor, es decir, dos instalaciones de JBoss y dos instalaciones de Tomcat en cada una de las computadoras, ya que esto fue muy útil en la etapa de aprendizaje y primera configuración, fue muy práctico tener todo el ambiente en una misma computadora armando un Cluster de topología vertical para poder confirmar que la configuración es la adecuada. Claramente, en cada computadora, una de las instalaciones de JBoss ofició de worker1 mientras que la otra hizo las veces de worker2. De la misma manera, una de las instalaciones de Tomcat fue de worker1 y la otra fue worker2. Posteriormente, al momento de armar el

Cluster con topología horizontal, se tomó para cada caso el worker1 de uno de los nodos, y el worker2 de otro de los nodos, es decir, no se tomó para armar el Cluster cada worker1 de las computadoras intervinientes, sino que se mantuvo la nomenclatura y se tomó el worker1 de una de las computadoras para worker1, y el worker2 de otra de las computadoras para worker2.

También cabe recordar en el contexto de esta sección, que siempre el sistema operativo es Ubuntu Linux.

JBoss AS: Cluster

Se define [home 1 de JBoss 6] como directorio de instalación de la instancia de JBoss que conforma el worker1, por ejemplo /opt/JBoss-6.1.0.Final, y [home 2 de JBoss 6] como directorio de instalación de la instancia de JBoss que conforma el worker2, por ejemplo /opt/JBoss-6.1.0.Final.2., independientemente de la computadora.

Si en worker1 se ejecuta:

```
[home 1 de JBoss 6]/bin/run.sh -c all -g qin -u 239.1.2.3 -b
worker1 -Djboss.messaging.ServerPeerID=1
-Djboss.service.binding.set=ports-default
```

... y en worker2 se ejecuta:

```
[home 2 de JBoss 6]/bin/run.sh -c all -g qin -u 239.1.2.3 -b
worker2 -Djboss.messaging.ServerPeerID=2
-Djboss.service.binding.set=ports-01
```

... se arma un Cluster. Explicación de las llamadas:

[home 1 de JBoss 6]/bin/run.sh: comando para ejecutar la instancia de JBoss 6; ídem que [home 2 de JBoss 6]/bin/run.sh
 -c all: perfil a utilizar; JBoss 6 provee los perfiles: all, default, jbossweb-standalone, minimal, standard; el perfil all es el recomendado por la bibliografía para trabajar con Clusters [JBoss AS IN ACTION]
 -g qin: nombre de fantasía del Cluster, con el que cada instancia de JBoss se asociará, que además sirve también para distinguir este Clusters de otros Clusters, si hubiere; en nuestro caso, el nombre es qin, como el de todo el proyecto
 -u 239.1.2.3: dirección IP para multicast, utilizada por los mecanismos de sincronización del Cluster, como el servicio de multicaste JGroups (<http://www.jgroups.org/>) y el sistema de mensajería Hornetq (<https://www.jboss.org/hornetq>) en que se basa el clustering provisto por JBoss

-b worker1: dirección IP en que atiende esta instancia en particular; se utiliza directamente worker1, para que se levante el IP definido para el alias worker1 en /etc/hosts; equivalente en el caso de -b worker2.

-Djboss.messaging.ServerPeerID=1: identificación del canal de mensajería que se le asigna al nodo; único para cada nodo; en este caso se toma el canal 1. En el caso de worker2 el canal es el 2.

-Djboss.service.binding.set=ports-default: set de puertos a tomar en la computadora en que corre el nodo; en este caso, el set por defecto, que determina http en el puerto 8080 y ajp en el puerto 8009. En el caso de -Djboss.service.binding.set=ports-01, se le indica a la instancia de JBoss que incremente el número de sus puertos en una centena (se determina la cantidad de centenas con el "01" de "ports-01"; en este caso, una centena); esto no es necesario cuando los nodos se encuentran en computadoras distintas, es decir cuando la topología es horizontal, pero igualmente decidimos utilizar siempre esta opción para el worker2, para de esta manera poder levantar todo en una sola computadora, si así fuera necesario, con los mismos comandos. Naturalmente, http queda en el puerto 8180 y ajp queda en el puerto 8109.

Nótese que de esta manera, los comandos para levantar el Cluster quedan escritos "en genérico", y sirven para cuanta combinación se determine, actualizando los valores de los IPs de worker1 y worker2 en el archivo /etc/hosts según lo que se necesite.

Ambos comandos descritos anteriormente, se complementan con la edición de respectivamente [home de JBoss 6]/server/all/deploy/jbossweb.sar/server.xml y de [home 2 de JBoss 6]/server/all/deploy/jbossweb.sar/server.xml. En la línea 33 existe:

```
<Engine name="JBoss.web" defaultHost="localhost">
```

... y se lo debe editar de la manera:

```
<Engine name="JBoss.web" defaultHost="localhost"
jvmRoute="worker1">
```

... y

```
<Engine name="JBoss.web" defaultHost="localhost"
jvmRoute="worker2">
```

... respectivamente. El valor del atributo `jvmRoute`, es el identificador con el que puede trabajar el balanceador de carga al ser configurado, como es nuestro caso, con `StickySession`, lo cual significa que una vez que un nodo atiende una petición para un usuario, dicho nodo atenderá todas las demás peticiones del mismo usuario; con este identificador, es que el balanceador de carga puede saber qué nodo (el nodo de `jvmRoute` igual a `worker1` o el nodo de `jvmRoute` igual a `worker2`) atiende a qué usuario.

Además, en todos los casos en `[home 1 de JBoss 6]/bin/run.conf` y `[home 1 de JBoss 6]/bin/run.conf` se anuló la configuración titulada "Sample JPDA settings for remote socket debugging", para evitar perder recursos al habilitar el debugging remoto.

Cabe mencionar, que el Cluster de JBoss que estamos manejando, con la política por defecto de elegibilidad de nodo máster, se comporta de manera tal que el primer nodo completamente iniciado termina actuando como nodo máster, y si algo le pasara a este nodo, el segundo nodo completamente iniciado pasará a ser master, y así siguiendo.

JBoss AS: standalone

Siempre que se necesitó trabajar con sólo un nodo de JBoss, se utilizó la configuración para worker1 vista anteriormente, a saber:

```
[home 1 de JBoss 6]/bin/run.sh -c all -g qin -u 239.1.2.3 -b  
worker1 -Djboss.messaging.ServerPeerID=1  
-Djboss.service.binding.set=ports-default
```

Este comando, no requiere a modo de complemento la ejecución del comando de worker2 para poder funcionar; es decir no es un comando que sólo tiene sentido al armar un Cluster. Simplemente, con este comando se puede levantar el nodo de worker1; al no ejecutar el comando de worker2, no se arma un Cluster; al ejecutar el comando de worker2, sí se arma el Cluster, tal como se explicó en la sección anterior.

Terracotta

La manera de ejecutar Terracotta es: ([home de Terracotta 3.6] es el directorio donde se instaló Terracotta 3.6.0 en esa computadora).

```
[home de Terracotta 3.6]/bin/start-tc-server.sh
```

Luego de levantar el servidor de Terracotta de esta manera, el mismo se quedará esperando a que se le conecten clientes. En nuestro caso, los clientes son instancias de Tomcat.

Tomcat: utilizando a Terracotta como sincronizador

Se define [home 1 de Tomcat 7] como directorio de instalación de la instancia de Tomcat que conforma el worker1, por ejemplo /opt/apache-tomcat-7.0.23, y [home 2 de Tomcat 7] como directorio de instalación de la instancia de Tomcat que conforma el worker2, por ejemplo /opt/apache-tomcat-7.0.23.2., independientemente de la computadora.

Antes de armar el Cluster con Tomcat, se debe editar el archivo `[home 2 de Tomcat 7]/conf/server.xml` para corregir los puertos de los servicios de http y ajp del nodo de worker2, para que éste no se solape con el nodo de worker1. Como en el caso de la instancia de worker2 de JBoss, se coloca el servicio de http en el puerto 8180 y el de ajp en el puerto 8109. También, en dicho archivo, en la línea 103, se debe incorporar como en el caso de JBoss el atributo `jvmRoute`, de manera tal que quede:

```
<Engine name="Catalina" defaultHost="localhost"
jvmRoute="worker1">
```

... en worker1 y

```
<Engine name="Catalina" defaultHost="localhost"
jvmRoute="worker2">
```

... en worker2.

Se levanta cada instancia del Cluster implementado con Tomcat, de esta manera:

```
[home 1 de Tomcat 7]/bin/startup.sh
```

```
[home 2 de Tomcat 7]/bin/startup.sh
```

También se aplica en este caso, la decisión de no utilizar recursos en debugging remoto, lo cual se puede ver respectivamente en `[home 1 de Tomcat 7]/bin/catalina.sh` y `[home 2 de Tomcat 7]/bin/catalina.sh`.

La conexión con Terracotta se configura en los archivos `[home de Tomcat 7]/conf/context.xml` y `[home 2 de Tomcat 7]/conf/context.xml`, a los que se incorpora la línea:

```
<Valve
className="org.terracotta.session.TerracottaTomcat70xSessionValve"
tcConfigUrl="terracotta:9510"/>
```

El valor `"terracotta:9510"`, refiere al IP de alias terracotta, anotado en el archivo `/etc/hosts`.

Como se puede apreciar, en el comando para levantar una instancia de Tomcat no incluimos parámetro alguno en particular para ayudar a armar el Cluster. Justamente, Terracotta trabaja sobre N instancias de Tomcat que corren standalone, y es Terracotta mismo el componente que incorpora las funcionalidades de sincronización y administración, logrando que esas N instancias, en nuestro caso dos, actúen como un Cluster organizado.

Tomcat: standalone

Como se explicó anteriormente, en realidad el Cluster armado con Terracotta + Tomcat está compuesto de instancias standalone de Tomcat, ya que la sincronización no la provee Tomcat sino Terracotta, por lo cual el comando para levantar un Tomcat standalone es tal como se vio anteriormente:

```
[home 1 de Tomcat 7]/bin/startup.sh
```

JMeter

JMeter se levanta de manera normal, ejecutando su script principal `jmeter.sh`.

A modo de particularidad, sólo cabe notar que al igual que los servidores que están también contruidos puramente en Java, es editó el script principal `jmeter.sh` para impedir el debugging remoto para ahorrar recursos.

7.2. Apéndice 2: Breve descripción de los scripts creados

Los scripts creados para realizar el presente trabajo, son en orden alfabético:

- `agregarEntradaEnRouter.sh`: incorporar en el router, la entrada para multicast udp del Cluster de JBoss.
- `backupearArchivosConfiguracion.sh`: antes de levantar Apache con su balanceador de carga `mod_proxy`, salvaguardar sus archivos de configuración.
- `bajarAmbiente.sh`: bajar Cluster, no importa cual.
- `bajarApacheCompleto.sh`: bajar solamente Apache, de manera completa, lo cual implica utilizar `graceful-stop` y recuperar los archivos de configuración salvaguardados.
- `bajarApache.sh`: bajar Apache, solamente.
- `bajarJboss.sh`: bajar cualquier instancia de JBoss.
- `bajarMysql.sh`: bajar el servicio de MySQL.
- `bajarTerracotta.sh`: bajar Terracotta.
- `bajarTomcat.sh`: bajar cualquier instancia de Tomcat.
- `compilarYDeployarQinweb.sh`: compilar y deployar el proyecto `qinweb`, tanto para y en JBoss como para y en Tomcat.
- `configurarProcesador.sh`: elevar la velocidad de todos los núcleos del CPU al máximo posible.
- `detectarIpConexion.sh`: detectar IP de la computadora.
- `editarArchivosConfiguracion.sh`: editar archivos de configuración de Apache y de su balanceador de carga `mod_proxy`.
- `jmeter.sh`: levantar Jmeter.
- `limpiarAmbiente.sh`: limpiar todos los archivos de log y temporales del entorno.
- `logearApache.sh`: mostrar en terminal, el log de Apache.
- `logearJboss.sh`: mostrar en terminal, el log de cualquier instancia de JBoss.
- `logearTerracotta.sh`: mostrar en terminal, el log de Terracotta.
- `logearTomcat.sh`: mostrar en terminal, el log de Tomcat.
- `manejarPermisos.sh`: para evitar complicaciones, dar todos los permisos a todos los servidores utilizados.
- `recargarConfiguracionApache.sh`: efectuar reload y `configtest` de la configuración de Apache.
- `reestablecerArchivosConfiguracion.sh`: recuperar los archivos de configuración salvaguardados de Apache y su balanceador de carga `mod_proxy`.
- `resetearBaseDeDatos.sh`: reconstruir la base de datos de pruebas, a su estado inicial.
- `resetearProcesador.sh`: colocar la velocidad de todos los núcleos del CPU en modo `onDemand`.

- `subirAmbiente.sh`: subir el Cluster entero, ya sea de JBoss o de Terracotta.
- `subirApacheCompleto.sh`: subir Apache y su balanceador de carga `mod_proxy`, recargando la configuración.
- `subirApache.sh`: solamente, subir Apache.
- `subirJboss.sh`: subir cualquier instancia de JBoss.
- `subirMysql.sh`: subir el servicio MySQL.
- `subirTerracotta.sh`: subir Terracotta.
- `subirTomcat.sh`: subir cualquier instancia de Tomcat
- `terracottaDevConsole.sh`: mostrar la consola de Terracotta.

Todos los scripts pueden ser ejecutados de manera independiente, o pueden llamarse automáticamente entre sí según se requiera. Todos los scripts toman parámetros por consola o en su defecto ejecutan sus parámetros por defecto.

A partir de que un script encapsula una tarea en particular, cada vez que se necesita de esa tarea en otro script se llama al script que encapsula esa tarea. Por ejemplo, el script `subirAmbiente.sh`, realiza llamadas a los scripts (en este orden):

- `bajarAmbiente.sh`
- `limpiarAmbiente.sh`
- `backupearArchivosConfiguracion.sh`
- `subirMysql.sh`
- `resetearBaseDeDatos.sh`
- `compilarYDeployarQinweb.sh`
- Si el Cluster se implementa con JBoss:
 - `agregarEntradaEnRouter.sh`
 - `subirJboss.sh` (dos veces)
- Si el Cluster se implementa con Terracotta + Tomcat:
 - `subirTerracotta.sh`
 - `subirTomcat.sh` (dos veces)
- `subirApacheCompleto.sh`
- `manejarPermisos.sh`

Todos los scripts ayudan a preservar los recursos de las computadoras en que se los corre: detectan el IP de la computadora, y si el mismo no es igual al IP de alias `basededatos` definido en `/etc/hosts`, bajan el servicio MySQL si es que está activo, porque si ese es el caso entonces es un desperdicio de recursos; el servicio MySQL sólo debe estar activo en la computadora cuyo IP es `basededatos`; lo mismo con `terracotta` y `apache`. Y además los scripts siempre elevan la velocidad de los núcleos del CPU.

7.3. Apéndice 3: Detalle del manejo de transacciones

El manejo de transacciones, está colocado en la capa de publicación de servicios, en nuestro esquema esto es en la capa de manager. Aquí se muestran dos ejemplos de la clase `AdministracionManagerImpl`; un borrado físico en la base de datos:

```
@Override
@Transactional
public void deleteMateria(Materia materia) throws Exception {
    materiaEAO.delete(materia);
}
```

... y una consulta a la base de datos

```
@Override
@Transactional(readonly = true, propagation =
Propagation.SUPPORTS)
public Alumno findAlumnoByUsuario(Usuario usuario) throws
Exception {
    return alumnoEAO.findById(usuario.getId());
}
```

Cómo se puede ver, el método que maneja el borrado se demarca con la annotation `@Transactional`, la cual como su nombre lo indica prepara al método para ser transaccional, es decir, provee a la ejecución del método una unidad de trabajo sobre la base de datos, con posibilidad de manejo de transacción.

En el método que implementa una consulta, en cambio, se puede ver la siguiente annotation: `@Transactional(readonly = true, propagation = Propagation.SUPPORTS)`. De esta manera, se demarca que el método interactúa con la base de datos, pero en particular de una manera de sólo lectura, y si no había una transacción activa al momento de invocar este método, no crea una, pero si había una activa, para optimizar se adhiere a esa transacción y la reutiliza también. Además, optimiza el acceso a la base de datos.

7.4. Apéndice 4: Diferencias en la implementación de la sincronización

Como se mencionó anteriormente, en la aplicación de pruebas hay una funcionalidad que debe estar sincronizada a nivel Cluster, que es la funcionalidad de sincronización de texto.

La misma, se presenta cuando más de un alumno resuelven de manera colaborativa un mismo trabajo práctico, previo intercambio del código de resolución compartida.

En ese contexto, el texto que vierten los alumnos que resuelven el trabajo práctico de manera colaborativa en el sistema, y que aún no ha sido almacenado en la base de datos, es sincronizado de manera centralizada y actualizado automáticamente cada cinco segundos, presentando en la sesión de cada uno de los alumnos vinculados a esa resolución el mismo texto actualizado; en otras palabras, a modo de ejemplo sólo para hacernos entender y salvando las distancias, una especie de Google Docs ahora renombrado Google Drive (http://es.wikipedia.org/wiki/Google_Drive).

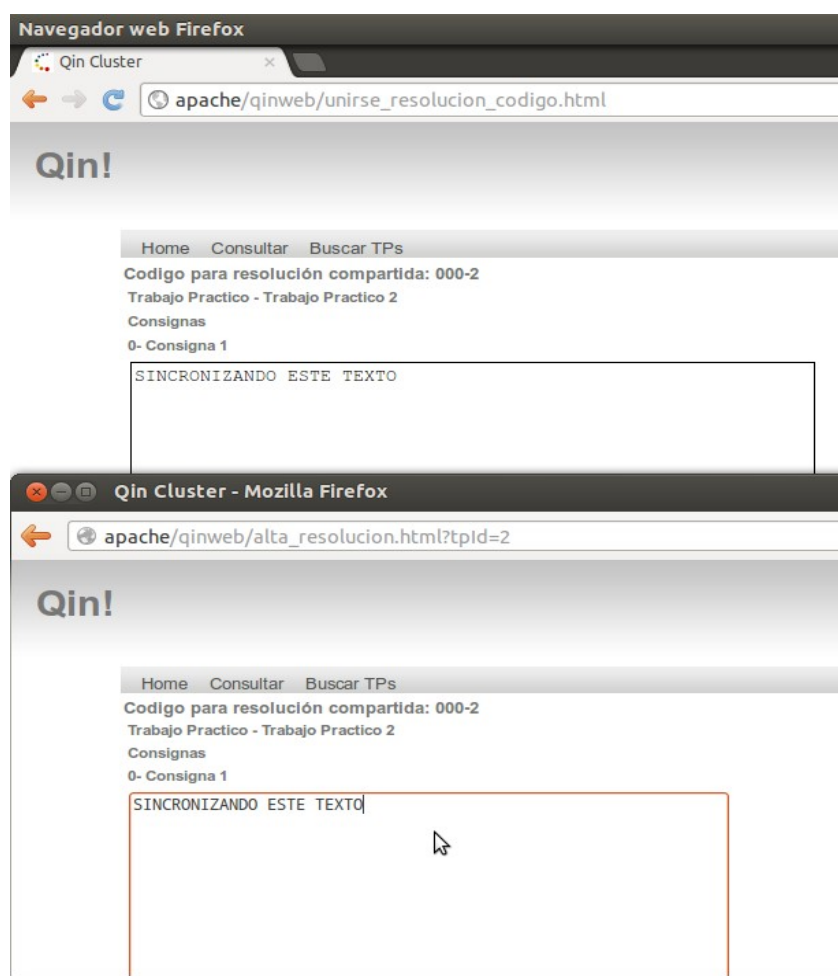


Ilustración 56: Sincronización de texto

La implementación de este servicio es en esencia la misma tanto para el Cluster estructurado con JBoss como para el Cluster estructurado con Terracotta + Tomcat: finalmente estamos hablando de una clase de sincronización de texto, que interactúa con un arreglo en que se almacenan todos los textos de todas las resoluciones colaborativas utilizadas por las sesiones de usuarios presentes y activas en el sistema. Dicho arreglo debe ser único en todo el Cluster y debe estar sincronizado.

Las pequeñas diferencias de las que se habla en varios sectores del informe, refieren a cómo se implementa con JBoss por un lado y con Terracotta + Tomcat por el otro dicha sincronización.

Por el lado de JBoss, la sincronización se implementa utilizando el componente de PojoCache de JBossCache (ver "7.5. Apéndice 5: Código de la clase *PojoCacheManagerImpl*, utilizada en el Cluster con JBoss"). Con dicho componente, se arma un cache de entidades no persistentes que es único y se sincroniza automáticamente a nivel Cluster. En este caso, las entidades no persistentes se reducen a solamente una: el arreglo con los textos a sincronizar. Para poder utilizar el PojoCache, se marcó la clase de sincronización de texto como un EJB Stateless.

Se presenta el código de la sincronización de texto en el Cluster de JBoss:

Interface:

```
@Local
public interface SincronizadorTexto {

    void activarTp(String id, String texto);

    void desactivarTp(String id);

    String actualizarTp(String id, LinkedList<Patch> patches);

    String obtenerTp(String id);

}
```

Implementación:

```
package com.qin.manager.colaboracion;

import java.util.LinkedList;

import javax.ejb.Stateless;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.qin.cache.pojo.manager.PojoCacheManager;
import com.qin.cache.pojo.manager.PojoCacheManagerImpl;
import com.qin.utils.HAUtils;
import com.qin.utils.diff_match_patch;
import com.qin.utils.diff_match_patch.Patch;

@Stateless
public class SincronizadorTextoImpl implements SincronizadorTexto
{

    private static Logger logger = LoggerFactory
        .getLogger(SincronizadorTextoImpl.class);

    private PojoCacheManager pojoCacheManager = null;

    public SincronizadorTextoImpl() {
    }

    @Override
    public void activarTp(String id, String texto) {
        try {
            pojoCacheManager = (PojoCacheManager) HAUtils
                .lookup(PojoCacheManagerImpl.JNDI_NAME);
            if (!pojoCacheManager.existsKey(id)) {
                pojoCacheManager.setValue(id, texto);
            }
        } catch (Throwable t) {
            t.printStackTrace();
            logger.error(t.getMessage());
        }
    }
}
```

```
@Override
public void desactivarTp(String id) {
    try {
        pojoCacheManager = (PojoCacheManager) HAUtils
            .lookup(PojoCacheManagerImpl.JNDI_NAME);
        pojoCacheManager.removeKey(id);
    } catch (Throwable t) {
        t.printStackTrace();
        logger.error(t.getMessage());
    }
}

@Override
public String actualizarTp(String id, LinkedList<Patch>
patches) {
    try {
        pojoCacheManager = (PojoCacheManager) HAUtils
            .lookup(PojoCacheManagerImpl.JNDI_NAME);
        String textoBase = pojoCacheManager.getValue(id);
        diff_match_patch dmp = new diff_match_patch();
        Object[] patch_apply = null;
        try {
            patch_apply = dmp.patch_apply(patches,
textoBase);
        } catch (Throwable t) {
            logger.debug("Error de patch_apply");
        }
        String nuevoTexto = null;
        try {
            nuevoTexto = (String) patch_apply[0];
        } catch (Throwable t) {
            logger.debug("Error de patch_apply");
            nuevoTexto = textoBase;
        }
        pojoCacheManager.setValue(id, nuevoTexto);
        return nuevoTexto;
    } catch (Throwable t) {
        t.printStackTrace();
        logger.error(t.getMessage());
        return null;
    }
}
```

```

@Override
public String obtenerTp(String id) {
    try {
        pojoCacheManager = (PojoCacheManager) HAUtils
            .lookup(PojoCacheManagerImpl.JNDI_NAME);
        String resultado = pojoCacheManager.getValue(id);
        return resultado;
    } catch (Throwable t) {
        t.printStackTrace();
        logger.error(t.getMessage());
        return null;
    }
}
}

```

Por el lado de Terracotta + Tomcat, la sincronización se implementa mediante Terracotta. En el código fuente se demarca que el arreglo con los textos a sincronizar es una instancia de un tipo especial de arreglo sincronizado que provee Terracotta, en su rol de administrador del Cluster que conforma. A partir de esa declaración, se puede usar el arreglo de manera libre donde se lo requiera; toda la sincronización a nivel Cluster queda del lado de Terracotta, así como en el caso de JBoss quedaba del lado de PojoCache. La clase de sincronización de texto no fue demarcada como un EJB Stateless como en el caso de JBoss, por el simple hecho de que Tomcat no soporta dicho componente; en su defecto, fue demarcada como servicio de Spring.

Se presenta el código de la sincronización de texto en el Cluster de Terracotta + Tomcat:

Interface:

```

package com.qin.manager.colaboracion;

import java.util.LinkedList;

import com.qin.utils.diff_match_patch.Patch;

public interface SincronizadorTexto {

    public void activarTp(String id, String texto);

    public void desactivarTp(String id);

    public String actualizarTp(String id, LinkedList<Patch>
patches);

    public String obtenerTp(String id);

}

```

Implementación:

```
package com.qin.manager.colaboracion;

import java.util.LinkedList;
import java.util.Map;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;
import org.terracotta.api.TerracottaClient;

import com.qin.utils.diff_match_patch;
import com.qin.utils.diff_match_patch.Patch;

@Service
public class SincronizadorTextoImpl implements SincronizadorTexto
{
    protected static Logger logger = LoggerFactory
        .getLogger(SincronizadorTextoImpl.class);

    private static Map<String, String> tpsActivos;

    static {
        tpsActivos = new
TerracottaClient("terracotta:9510").getToolkit().getMap(
        "myStaticMap");
    }

    public void activarTp(String id, String texto) {
        if (tpsActivos.get(id) == null) {
            tpsActivos.put(id, texto);
        }
    }

    public void desactivarTp(String id) {
        tpsActivos.remove(id);
    }

    public String actualizarTp(String id, LinkedList<Patch>
patches) {
        String textoBase = tpsActivos.get(id);
        diff_match_patch dmp = new diff_match_patch();
        Object[] patch_apply = dmp.patch_apply(patches,
textoBase);
        String nuevoTexto = (String) patch_apply[0];
        tpsActivos.put(id, nuevoTexto);
        return nuevoTexto;
    }
}
```

```
public String obtenerTp(String id) {  
    return tpsActivos.get(id);  
}  
  
}
```

Como se puede apreciar, si bien la implementación es distinta, en esencia, se tiene una clase cuyo comportamiento real está dado por los métodos:

```
void activarTp(String id, String texto);  
void desactivarTp(String id);  
String actualizarTp(String id, LinkedList<Patch> patches);  
String obtenerTp(String id);
```

... cuya implementación consta de consultar o actualizar un arreglo sincronizado externamente y desplegado automáticamente en todo el Cluster, y en la que todo lo demás es infraestructura auxiliar.

Debido a esta conclusión, es que se sostiene en varias secciones del presente trabajo, que la diferencia en la implementación de la sincronización es insignificante y no incide negativamente sobre las comparaciones entre Clusters y nodos individuales.

7.5. Apéndice 5: Código de la clase *PojoCacheManagerImpl*, utilizada en el Cluster con JBoss

El PojoCache se configura mediante el archivo `qinejb/src/main/resources/META-INF/JBoss-service.xml`, cuyo contenido es:

```
<?xml version="1.0" encoding="UTF-8"?>

<server>
  <mbean name="qinejb:service=PojoCacheManager-Controller"
    code="org.JBoss.ha.singleton.HASingletonController">
    <depends>qinejb:service=PojoCacheManager</depends>
    <attribute name="HAPartition">
      <inject bean="HAPartition" />
    </attribute>
    <attribute
name="TargetName">qinejb:service=PojoCacheManager</attribute>
    <attribute
name="TargetStartMethod">startSingleton</attribute>
    <attribute
name="TargetStopMethod">stopSingleton</attribute>
    </mbean>
</server>
```

Como puede verse en `JBoss-service.xml`, se definen el método para iniciar el servicio, de nombre `startSingleton`, y el método para parar el servicio, de nombre `stopSingleton`. Esos métodos se implementan en:

Interface remota

```
package com.qin.cache.pojo.manager;

import javax.ejb.Remote;
import javax.naming.NamingException;

@Remote
public interface PojoCacheManager {

    boolean isMasterNode();

    void create() throws Exception;

    void start() throws Exception;

    void stop();

    void destroy();

    void startSingleton() throws NamingException;

    void stopSingleton() throws NamingException;

    boolean existsKey(String key);

    void setValue(String key, String value);

    void removeKey(String key);

    String getValue(String key);
}
```

Implementación: se demarca en negrita, el POJO cacheado, en este caso el arreglo con los textos de resolución colaborativa.

```
package com.qin.cache.pojo.manager;

import java.util.HashMap;
import java.util.Map;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.JBoss.ejb3.annotation.Management;
import org.JBoss.ejb3.annotation.Service;
import org.JBoss.naming.Util;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Service(objectName = "qinejb:service=PojoCacheManager")
@Management(PojoCacheManager.class)
public class PojoCacheManagerImpl implements PojoCacheManager {

    protected static Logger logger = LoggerFactory
        .getLogger(PojoCacheManagerImpl.class);

    public static final String JNDI_NAME = "qinear-
1.0/PojoCacheManagerImpl/remote";

    private boolean masterNode = false;

    private PojoCacheManager reference;

    private Map<String, String> pojoCache = new HashMap<String,
String>();

    @Override
    public boolean isMasterNode() {
        return masterNode;
    }

    @Override
    public void create() throws Exception {
    }
```

```
@Override
public void start() throws Exception {
    try {
        // Remove from local JNDI until started as a
singleton,
        // but keep the reference locally so it could be
restored later.
        Context ctx = new InitialContext();
        reference = (PojoCacheManager)
ctx.lookup(JNDI_NAME);
        Util.unbind(ctx, JNDI_NAME);
    } catch (Throwable t) {
        t.printStackTrace();
        logger.debug(t.getMessage());
    }
}

@Override
public void stop() {
}

@Override
public void destroy() {
}

/**
 * Called by HASingletonController on singleton start
 */
@Override
public void startSingleton() throws NamingException {
    try {
        masterNode = true;
        // Rebind to JNDI when started as singleton
        Context ctx = new InitialContext();
        Util.rebind(ctx, JNDI_NAME, reference);
    } catch (Throwable t) {
        t.printStackTrace();
        logger.debug(t.getMessage());
        masterNode = false;
    }
}
```

```
/**
 * Called by HASingletonController on singleton stop
 */
@Override
public void stopSingleton() throws NamingException {
    try {
        logger.debug("stopSingleton");
        masterNode = false;
        // Unbind from local JNDI when stopped
        Context ctx = new InitialContext();
        Util.unbind(ctx, JNDI_NAME);
    } catch (Throwable t) {
        t.printStackTrace();
        logger.debug(t.getMessage());
        masterNode = false;
    }
}

@Override
public boolean existsKey(String key) {
    try {
        return pojoCache.containsKey(key);
    } catch (Throwable t) {
        t.printStackTrace();
        logger.error(t.getMessage());
        return false;
    }
}

@Override
public void setValue(String key, String value) {
    try {
        if (value == null) {
            value = "";
        }
        pojoCache.put(key, value);
    } catch (Throwable t) {
        t.printStackTrace();
        logger.error(t.getMessage());
    }
}

@Override
public void removeKey(String key) {
    try {
        pojoCache.remove(key);
    } catch (Throwable t) {
        t.printStackTrace();
        logger.error(t.getMessage());
    }
}
```

```

@Override
public String getValue(String key) {
    try {
        String retorno = pojoCache.get(key);
        if (retorno == null) {
            retorno = "";
        }
        return retorno;
    } catch (Throwable t) {
        t.printStackTrace();
        logger.error(t.getMessage());
        return null;
    }
}
}

```

El comportamiento a nivel Cluster, es el siguiente: se deploja tanto el archivo JBoss-service.xml en todos los nodos del Cluster como el código fuente conteniendo la interface PojoCacheManager y la clase PojoCacheManagerImpl, y cuando esto ocurre en todos los nodos se ejecuta el método start definido en la interface remota. Dicho método, lo que hace es desconectar del repositorio JNDI el servicio de sincronización de Pojos, pero lo deja en una referencia local para poder ser utilizado posteriormente. En el nodo máster, se procesa el archivo JBoss-service.xml, y se ejecuta el método definido para comenzar el servicio, en este caso, startSingleton (es solamente un nombre; este artefacto no es estrictamente hablando un singleton, pero conceptualmente funciona como "un singleton a nivel Cluster", lo cual no tiene soporte nativo). Éste último método, lo que hace es tomar la referencia local del servicio anteriormente eliminada del repositorio JNDI, y lo vuelve a registrar, por ende a partir de ese momento cuando cualquier nodo quiera comunicarse con el servicio, el único registrado por lo cual el único que atenderá, será aquel cuya implementación es la del nodo máster, logrando de esta manera una sincronización centralizada a nivel Cluster de los Pojos.

Si algo ocurriera con el nodo máster y éste dejara de funcionar, aplicando los mismos procedimientos, se activará el servicio en el nodo iniciado en segundo lugar, y así siguiendo.

De esta manera, se logra armar un cache sincronizado de textos de los trabajos prácticos que se resuelven de manera colaborativa, que es único en todo el Cluster y que puede ser accedido por cualquier nodo del mismo.

[POJO FAQ] [POJO TUTORIAL] [POJO USER GUIDE] [CLUSTERING GUIDE 5.1 CH 4]

7.6. Apéndice 6: Ejecución de la combinación de casos de pruebas manuales

JBoss

En el caso de JBoss, se provee a continuación evidencia de una corrida de las características que permiten comprobar las pruebas manuales.

- Se levanta el Cluster; se comprueba que se haya armado correctamente.

Worker1 activándose y levantando el PojoCacheManager: cabe destacar, que como es el nodo que se levanta primero, se inicia en este nodo con `startSingleton` el servicio de sincronización (ver "7.4. Apéndice 4: Diferencias en la implementación de la sincronización" y "7.5. Apéndice 5: Código de la clase `PojoCacheManagerImpl`, utilizada en el Cluster con JBoss").

```
04:25:13,773 INFO [JBossASKernel] JBoss.j2ee:ear=qinear-
1.0.ear,jar=qinejb-1.0.jar,name=PojoCacheManagerImpl,service=EJB3;
Required: Described
04:25:13,773 INFO [JBossASKernel] and supplies:
04:25:13,773 INFO [JBossASKernel]
Class:com.qin.cache.pojo.manager.PojoCacheManager
04:25:13,773 INFO [JBossASKernel] jndi:PojoCacheManagerImpl
04:25:13,774 INFO [JBossASKernel] jndi:qinear-
1.0/PojoCacheManagerImpl/remote
04:25:13,774 INFO [JBossASKernel] jndi:qinear-
1.0/PojoCacheManagerImpl/remote-
com.qin.cache.pojo.manager.PojoCacheManager
04:25:13,774 INFO [JBossASKernel] Installing
bean(qinejb:service=PojoCacheManager) into kernel
04:25:13,780 INFO [STDOUT] PojoCacheManagerImpl: create
04:25:13,783 INFO [EJBContainer] STARTED EJB:
com.qin.cache.pojo.manager.PojoCacheManagerImpl ejbName:
PojoCacheManagerImpl
04:25:14,353 INFO [JndiSessionRegistrarBase] Binding the
following Entries in Global JNDI:
```

```
qinear-1.0/PojoCacheManagerImpl/remote - EJB3.x Default Remote
Business Interface
qinear-1.0/PojoCacheManagerImpl/remote-
com.qin.cache.pojo.manager.PojoCacheManager - EJB3.x Remote
Business Interface
```

```
04:25:14,388 WARN [TimerServiceContainer] EJBTHREE-2193: using
deprecated TimerServiceFactory for restoring timers
04:25:14,741 INFO [STDOUT] PojoCacheManagerImpl: start
04:25:14,806 INFO [STDOUT] PojoCacheManagerImpl: startSingleton
04:25:14,843 INFO [TomcatDeployment] deploy, ctxPath=/qinweb
```

```
04:25:15,059 INFO [TransactionManagerFactory] Using a batchMode
transaction manager
04:25:15,175 INFO [JGroupsTransport] Starting JGroups Channel
04:25:15,175 INFO [JChannel] JGroups version: 2.11.0.GA
```

Worker2 activándose y levantando el PojoCacheManager, sin ejecutar startSingleton.

```
04:25:37,561 INFO [JBossASKernel] JBoss.j2ee:ear=qinear-
1.0.ear,jar=qinejb-1.0.jar,name=PojoCacheManagerImpl,service=EJB3;
Required: Described
04:25:37,562 INFO [JBossASKernel] and supplies:
04:25:37,562 INFO [JBossASKernel]
Class:com.qin.cache.pojo.manager.PojoCacheManager
04:25:37,562 INFO [JBossASKernel] jndi:PojoCacheManagerImpl
04:25:37,562 INFO [JBossASKernel] jndi:qinear-
1.0/PojoCacheManagerImpl/remote
04:25:37,562 INFO [JBossASKernel] jndi:qinear-
1.0/PojoCacheManagerImpl/remote-
com.qin.cache.pojo.manager.PojoCacheManager
04:25:37,562 INFO [JBossASKernel] Installing
bean(qinejb:service=PojoCacheManager) into kernel
04:25:37,567 INFO [STDOUT] PojoCacheManagerImpl: create
04:25:37,570 INFO [EJBContainer] STARTED EJB:
com.qin.cache.pojo.manager.PojoCacheManagerImpl ejbName:
PojoCacheManagerImpl
04:25:37,594 INFO [JndiSessionRegistrarBase] Binding the
following Entries in Global JNDI:
```

```
qinear-1.0/PojoCacheManagerImpl/remote - EJB3.x Default Remote
Business Interface
qinear-1.0/PojoCacheManagerImpl/remote-
com.qin.cache.pojo.manager.PojoCacheManager - EJB3.x Remote
Business Interface
```

```
04:25:37,631 WARN [TimerServiceContainer] EJBTHREE-2193: using
deprecated TimerServiceFactory for restoring timers
04:25:37,706 INFO [STDOUT] PojoCacheManagerImpl: start
04:25:37,775 INFO [TomcatDeployment] deploy, ctxPath=/qinweb
04:25:37,902 INFO [TransactionManagerFactory] Using a batchMode
transaction manager
04:25:37,921 INFO [JGroupsTransport] Starting JGroups Channel
04:25:37,921 INFO [JChannel] JGroups version: 2.11.0.GA
```


Worker1 detecta que se inicia worker2, lo toma para armar un Cluster, y arranca definitivamente el PojoCacheManager, ya que ya se ha determinado que la instancia que queda sincronizando a nivel Cluster es la de worker1.

```
2012-10-17 04:25:27,544 INFO
[org.JBoss.ha.framework.server.ClusterPartition.lifecycle.qin]
(Incoming-5,null) New cluster view for partition qin (id: 1,
delta: 1, merge: false) : [worker1:1099, worker2:1199]
2012-10-17 04:25:27,548 INFO
[org.JBoss.ha.core.framework.server.DistributedReplicantManagerImp
l.qin] (AsynchViewChangeHandler Thread) I am (worker1:1099)
received membershipChanged event:
2012-10-17 04:25:27,549 INFO
[org.JBoss.ha.core.framework.server.DistributedReplicantManagerImp
l.qin] (AsynchViewChangeHandler Thread) Dead members: 0 ([])
2012-10-17 04:25:27,548 INFO
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(Incoming-5,null) Received new cluster view: [worker1:1099|1]
[worker1:1099, worker2:1199]
2012-10-17 04:25:27,552 INFO
[org.JBoss.ha.core.framework.server.DistributedReplicantManagerImp
l.qin] (AsynchViewChangeHandler Thread) New Members : 1
([worker2:1199])
2012-10-17 04:25:27,552 INFO
[org.JBoss.ha.core.framework.server.DistributedReplicantManagerImp
l.qin] (AsynchViewChangeHandler Thread) All Members : 2
([worker1:1099, worker2:1199])
2012-10-17 04:25:28,239 INFO
[org.hornetq.core.server.cluster.impl.BridgeImpl] (Thread-1
(group:HornetQ-server-threads1758530532-754513616)) Connecting
bridge sf.my-cluster.d13c5b06-182b-11e2-a756-68a3c49b1ef7 to its
destination [bac837c6-182b-11e2-aedb-68a3c49b1ef7]
2012-10-17 04:25:28,658 INFO
[org.hornetq.core.server.cluster.impl.BridgeImpl] (Thread-1
(group:HornetQ-server-threads1758530532-754513616)) Bridge sf.my-
cluster.d13c5b06-182b-11e2-a756-68a3c49b1ef7 is connected
[bac837c6-182b-11e2-aedb-68a3c49b1ef7-> sf.my-cluster.d13c5b06-
182b-11e2-a756-68a3c49b1ef7]
2012-10-17 04:25:38,028 INFO
[org.JBoss.ha.core.framework.server.CoreGroupCommunicationService.
lifecycle] (Incoming-18,null) New cluster view for partition null
(id: 1, delta: 1, merge: false) : [worker1:1099, worker2:1199]
2012-10-17 04:25:38,028 INFO
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(Incoming-18,null) Received new cluster view: [worker1:1099|1]
[worker1:1099, worker2:1199]
2012-10-17 04:25:39,577 INFO
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(Incoming-12,null) Received new cluster view: [worker1:1099|1]
[worker1:1099, worker2:1199]
```

```
(WorkerThread#0[192.168.1.107:49792]) PojoCacheManagerImpl:
existsKey
(WorkerThread#0[192.168.1.107:49793]) PojoCacheManagerImpl:
setValue
(WorkerThread#0[192.168.1.107:49794]) PojoCacheManagerImpl:
getValue
```

- Se ingresa con el navegador Mozilla Firefox con usuario1.



Ilustración 57: Usuario1 en Mozilla Firefox

- Se confirma qué nodo atiende a usuario1; es el worker2.

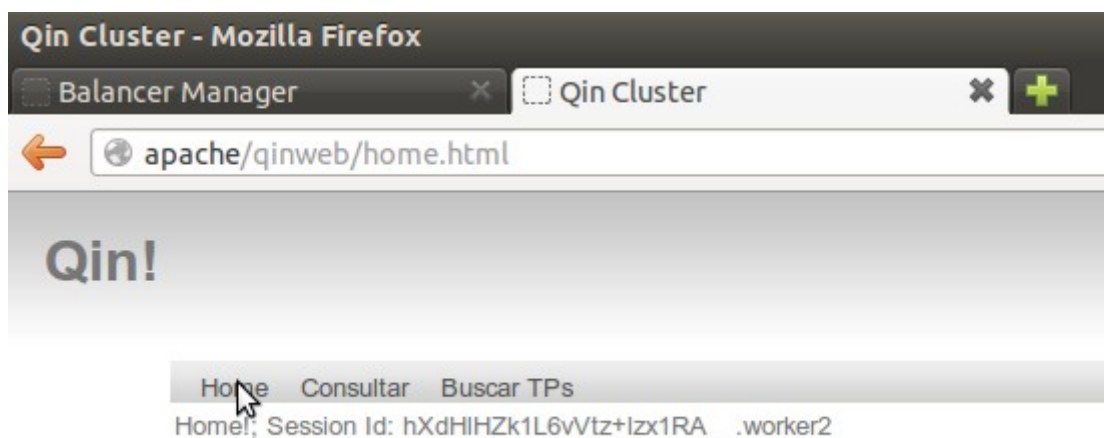


Ilustración 58: Usuario1 en worker2

- Se ingresa con el navegador Chromium con usuario2.



Ilustración 59: Usuario2 en Chromium

- Se confirma qué nodo atiende a usuario2; es el worker1.

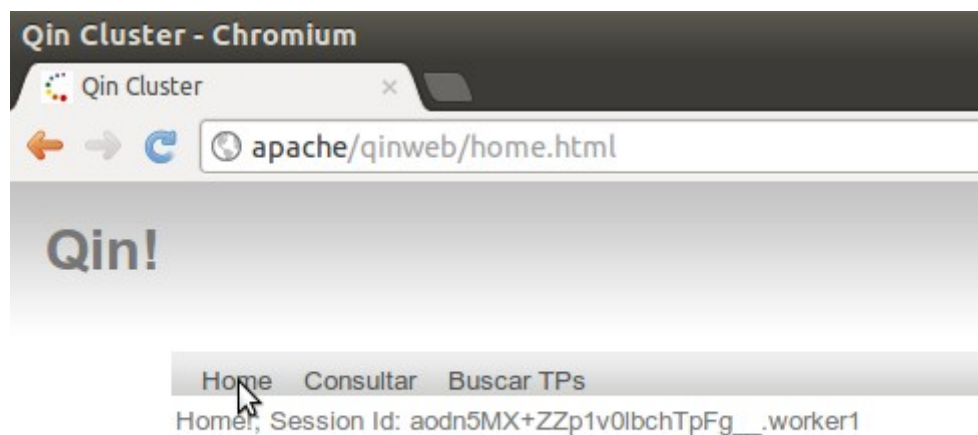


Ilustración 60: Usuario2 en worker1

- Se verifica que el sincronizador está trabajando.

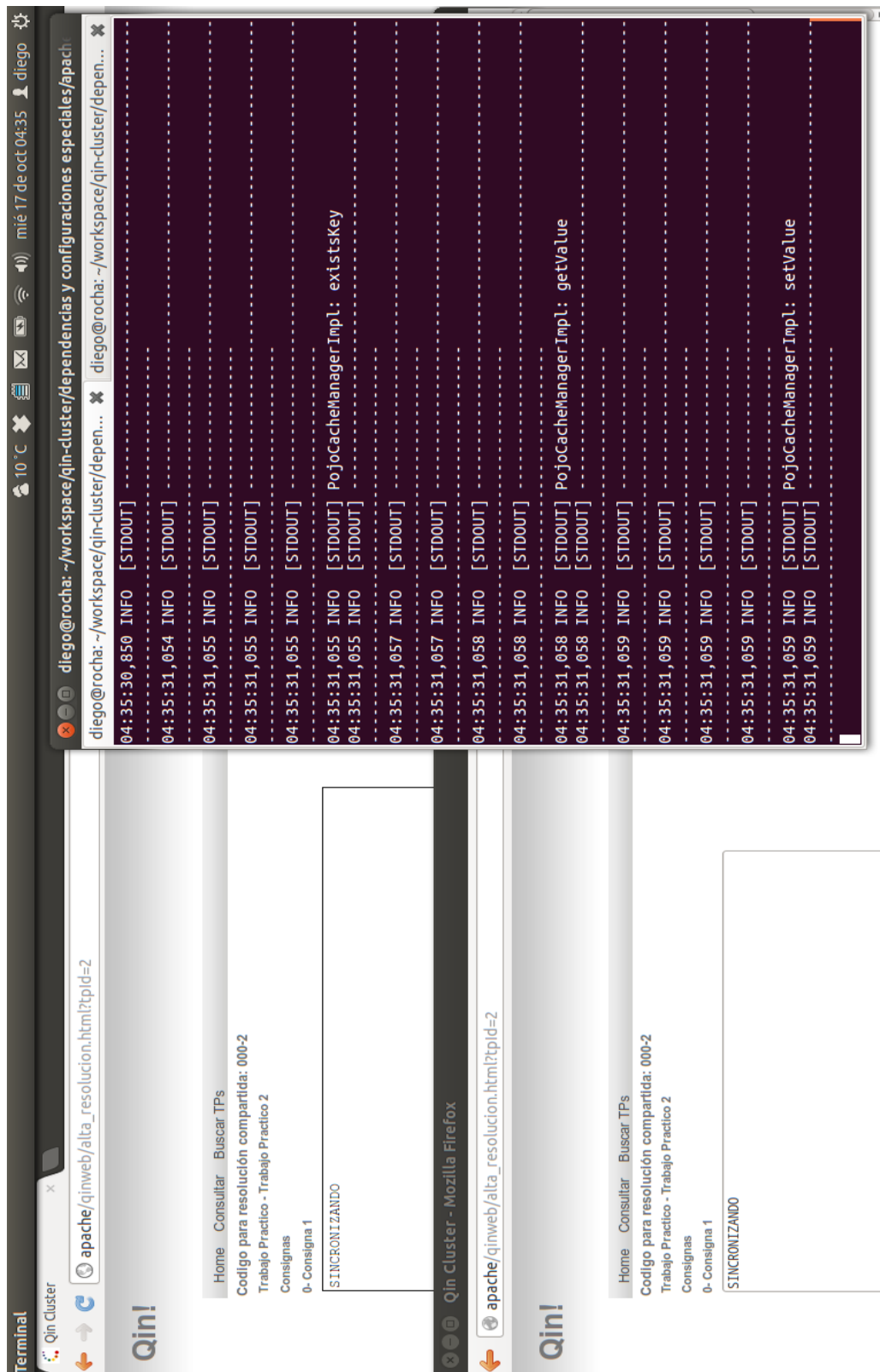


Ilustración 61: Sincronizador trabajando

- Se baja worker1; comprobar que la sincronización continúa; se superan las pruebas manuales *"Desconexión de un nodo en esquema colaborativo"* y *"Disponibilidad del sistema"*.

Worker1 bajando y desconectando el PojoCacheManager.

```

2012-10-17 04:36:03,394 INFO
[org.JBoss.system.server.jmx.JMXKernel] (Thread-30) Server exit
called, exiting the JVM now!
2012-10-17 04:36:03,409 INFO [STDOUT] (Thread-1) Posting Shutdown
Request to the server...
2012-10-17 04:36:03,409 INFO
[org.JBoss.bootstrap.impl.base.server.AbstractServer] (Thread-31)
Stopping: JBossAS [6.1.0.Final "Neo"]
2012-10-17 04:36:03,410 INFO
[org.apache.coyote.http11.Http11Protocol] (Thread-31) Pausando
Coyote HTTP/1.1 en puerto http-worker1%2F192.168.1.107-8080
2012-10-17 04:36:03,411 INFO
[org.apache.coyote.http11.Http11Protocol] (Thread-31) Parando
Coyote HTTP/1.1 en puerto http-worker1%2F192.168.1.107-8080
2012-10-17 04:36:03,437 INFO [org.apache.coyote ajp.AjpProtocol]
(Thread-31) Pausando Coyote AJP/1.3 en ajp-
worker1%2F192.168.1.107-8009
2012-10-17 04:36:03,438 INFO [org.apache.coyote ajp.AjpProtocol]
(Thread-31) Parando Coyote AJP/1.3 en ajp-worker1%2F192.168.1.107-
8009
...
04:36:03,486 INFO [service] Restored bootstrap log handlers
04:36:03,501 INFO [TomcatDeployment] undeploy, ctxPath=/qinweb
04:36:03,505 INFO [[/qinweb]] Destroying Spring FrameworkServlet
'qinweb'
04:36:03,510 INFO [SessionFactoryImpl] closing
04:36:03,518 INFO [DefaultCacheContainerFactory] Stopped
"com.qin.entity.Materia" cache from "hibernate" container
04:36:03,525 INFO [DefaultCacheContainerFactory] Stopped "local-
query" cache from "hibernate" container
04:36:03,530 INFO [DefaultCacheContainerFactory] Stopped
"org.hibernate.cache.UpdateTimestampsCache" cache from "hibernate"
container
04:36:03,544 INFO [DefaultCacheContainerFactory] Stopped
"//localhost/qinweb" cache from "web" container
04:36:03,565 INFO [STDOUT] PojoCacheManagerImpl: stopSingleton
04:36:03,573 INFO [STDOUT] PojoCacheManagerImpl: stop
04:36:03,612 INFO [EJBContainer] STOPPED EJB:
com.qin.cache.pojo.manager.PojoCacheManagerImpl ejbName:
PojoCacheManagerImpl
04:36:03,612 INFO [STDOUT] PojoCacheManagerImpl: destroy
04:36:03,618 INFO [SessionSpecContainer] Stopping
JBoss.j2ee:ear=qinear-1.0.ear,jar=qinejb-
1.0.jar,name=SincronizadorTextoImpl,service=EJB3

```

```
04:36:03,620 INFO [EJBContainer] STOPPED EJB:
com.qin.manager.colaboracion.SincronizadorTextoImpl ejbName:
SincronizadorTextoImpl
04:36:03,641 INFO [TomcatDeployment] undeploy, ctxPath=/
04:36:03,699 INFO [ConnectionFactoryBindingService] Unbound
ConnectionFactory
'JBoss.jca:service=ConnectionFactoryBinding,name=JmsXA' from JNDI
name 'java:JmsXA'
04:36:03,707 INFO [PersistenceUnitDeployment] Stopping
persistence unit persistence.unit:unitName=JBoss-ejb3-
timerservice-mk2.jar#timerdb
04:36:03,707 INFO [SessionFactoryImpl] closing
04:36:03,707 INFO [SessionFactoryObjectFactory] Unbinding factory
from JNDI name: persistence.unit:unitName=JBoss-ejb3-timerservice-
mk2.jar#timerdb
04:36:03,708 INFO [NamingHelper] JNDI InitialContext properties:
{java.naming.factory.initial=org.jnp.interfaces.NamingContextFacto
ry,
java.naming.factory.url.pkgs=org.JBoss.naming:org.jnp.interfaces}
04:36:03,709 INFO [SessionFactoryObjectFactory] Unbound factory
from JNDI name: persistence.unit:unitName=JBoss-ejb3-timerservice-
mk2.jar#timerdb
04:36:03,710 INFO [ConnectionFactoryBindingService] Unbound
ConnectionFactory
'JBoss.jca:service=DataSourceBinding,name=DefaultDS' from JNDI
name 'java:DefaultDS'
04:36:04,080 INFO [HypersonicDatabase] Database standalone closed
clean
04:36:04,082 INFO [QuartzScheduler] Scheduler
JBossQuartzScheduler_$_NON_CLUSTERED shutting down.
04:36:04,083 INFO [QuartzScheduler] Scheduler
JBossQuartzScheduler_$_NON_CLUSTERED paused.
04:36:04,083 INFO [QuartzScheduler] Scheduler
JBossQuartzScheduler_$_NON_CLUSTERED shutdown complete.
04:36:04,084 WARN [ComponentDeploymentContext] Removing name on
null names: JBoss.naming:application=quartz-ra
04:36:04,088 WARN [ComponentDeploymentContext] Removing name on
null names: JBoss.naming:application=mail-ra
04:36:04,091 INFO [HornetQResourceAdapter] HornetQ resource
adapter stopped
04:36:04,092 WARN [ComponentDeploymentContext] Removing name on
null names: JBoss.naming:application=jms-ra
04:36:04,096 WARN [ComponentDeploymentContext] Removing name on
null names: JBoss.naming:application=JBoss-xa-jdbc
04:36:04,099 WARN [ComponentDeploymentContext] Removing name on
null names: JBoss.naming:application=JBoss-local-jdbc
04:36:04,111 INFO [TomcatDeployment] undeploy, ctxPath=/invoker
04:36:04,118 INFO [TomcatDeployment] undeploy, ctxPath=/juddi
04:36:04,121 INFO [RegistryServlet] jUDDI Stopping: Cleaning up
existing resources.
```

04:36:04,151 INFO [JBossSatx] ARJUNA-32018 Destroying TransactionManagerService
04:36:04,157 INFO [JBossSatx] ARJUNA-32014 Stopping transaction recovery manager
04:36:04,203 INFO [DefaultCacheContainerFactory] Stopped "distributed-tree" cache from "ha-partition" container
04:36:04,214 INFO [qin] Stopping partition qin
04:36:04,223 INFO [DefaultCacheContainerFactory] Stopped "distributed-state" cache from "ha-partition" container
04:36:04,224 INFO [qin] Partition qin stopped.
04:36:04,224 INFO [qin] Partition qin destroyed.
04:36:04,386 INFO [BridgeImpl] Bridge sf.my-cluster.d13c5b06-182b-11e2-a756-68a3c49b1ef7 being stopped
04:36:04,390 INFO [BridgeImpl] stopped bridge sf.my-cluster.d13c5b06-182b-11e2-a756-68a3c49b1ef7
04:36:04,519 INFO [HornetQServerImpl] HornetQ Server version 2.2.5.Final (HQ_2_2_5_FINAL_AS7, 121) [bac837c6-182b-11e2-aedb-68a3c49b1ef7] stopped
04:36:04,608 INFO [DefaultCacheContainerRegistry] Unbinding hibernate cache container from java:CacheManager/entity
04:36:04,622 INFO [JGroupsTransport] Disconnecting and closing JGroups Channel
04:36:04,943 INFO [JGroupsTransport] Stopping the RpcDispatcher
04:36:04,946 INFO [JGroupsTransport] Disconnecting and closing JGroups Channel
04:36:05,252 INFO [JGroupsTransport] Stopping the RpcDispatcher
04:36:05,255 INFO [JGroupsTransport] Disconnecting and closing JGroups Channel
04:36:05,582 INFO [JGroupsTransport] Stopping the RpcDispatcher
04:36:05,586 INFO [poa] POA Naming destroyed
04:36:05,591 INFO [poa] POA TPOA destroyed
04:36:05,592 INFO [poa] POA PPOA destroyed
04:36:05,592 INFO [MailService] Mail service 'java:/Mail' removed from JNDI
04:36:05,600 INFO [JMXConnector] JMXConnector stopped
04:36:05,613 INFO [orb] prepare ORB for shutdown...
04:36:05,613 INFO [orb] ORB going down...
04:36:05,618 INFO [poa] POA IR destroyed
04:36:05,626 INFO [poa] POA RootPOA destroyed
04:36:05,628 INFO [poa] POA OTS destroyed
04:36:05,628 INFO [poa] POA OTSResources destroyed
04:36:05,631 INFO [orb] ORB shutdown complete
04:36:05,631 INFO [orb] ORB run, exit
04:36:05,631 INFO [iiop] Listener exited
04:36:05,706 INFO [SnmpAgentService] SNMP agent stopped
04:36:07,294 INFO [AbstractServer] Stopped: JBossAS [6.1.0.Final "Neo"] in 3s:884ms

Worker2 activando su instancia de PojoCacheManager, para que ahora ésta pueda ser la instancia que sincroniza todo el Cluster, y registrando que el Cluster perdió un nodo.

```
2012-10-17 04:36:03,586 INFO [STDOUT] (AsynchKeyChangeHandler Thread) PojoCacheManagerImpl: startSingleton
```

```
2012-10-17 04:36:04,418 WARN
```

```
[org.hornetq.core.protocol.core.impl.RemotingConnectionImpl]
(Thread-4 (group:HornetQ-client-global-threads-1602965727))
Connection failure has been detected: The connection was
disconnected because of server shutdown [code=4]
```

```
2012-10-17 04:36:04,419 WARN
```

```
[org.hornetq.core.server.cluster.impl.BridgeImpl] (Thread-4
(group:HornetQ-client-global-threads-1602965727)) sf.my-
cluster.bac837c6-182b-11e2-aedb-68a3c49b1ef7::Connection failed
before reconnect : HornetQException[errorCode=4 message=The
connection was disconnected because of server shutdown]
    at
org.hornetq.core.client.impl.ClientSessionFactoryImpl$Channel0Hand
ler$1.run(ClientSessionFactoryImpl.java:1262) [:6.1.0.Final]
    at
org.hornetq.utils.OrderedExecutorFactory$OrderedExecutor$1.run(Ord
eredExecutorFactory.java:100) [:6.1.0.Final]
    at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecut
or.java:1110) [:1.6.0_24]
    at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecu
tor.java:603) [:1.6.0_24]
    at java.lang.Thread.run(Thread.java:679) [:1.6.0_24]
```

```
2012-10-17 04:36:04,422 WARN
```

```
[org.hornetq.core.protocol.core.impl.RemotingConnectionImpl]
(Thread-5 (group:HornetQ-client-global-threads-1602965727))
Connection failure has been detected: The connection was
disconnected because of server shutdown [code=4]
```

```
2012-10-17 04:36:04,930 INFO
```

```
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(Incoming-9,null) Received new cluster view: [worker2:1199|2]
[worker2:1199]
```

```
2012-10-17 04:36:05,251 INFO
```

```
[org.JBoss.ha.framework.server.ClusterPartition.lifecycle.qin]
(Incoming-11,null) New cluster view for partition qin (id: 2,
delta: -1, merge: false) : [worker2:1199]
```

```
2012-10-17 04:36:05,251 INFO
```

```
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(Incoming-11,null) Received new cluster view: [worker2:1199|2]
[worker2:1199]
```

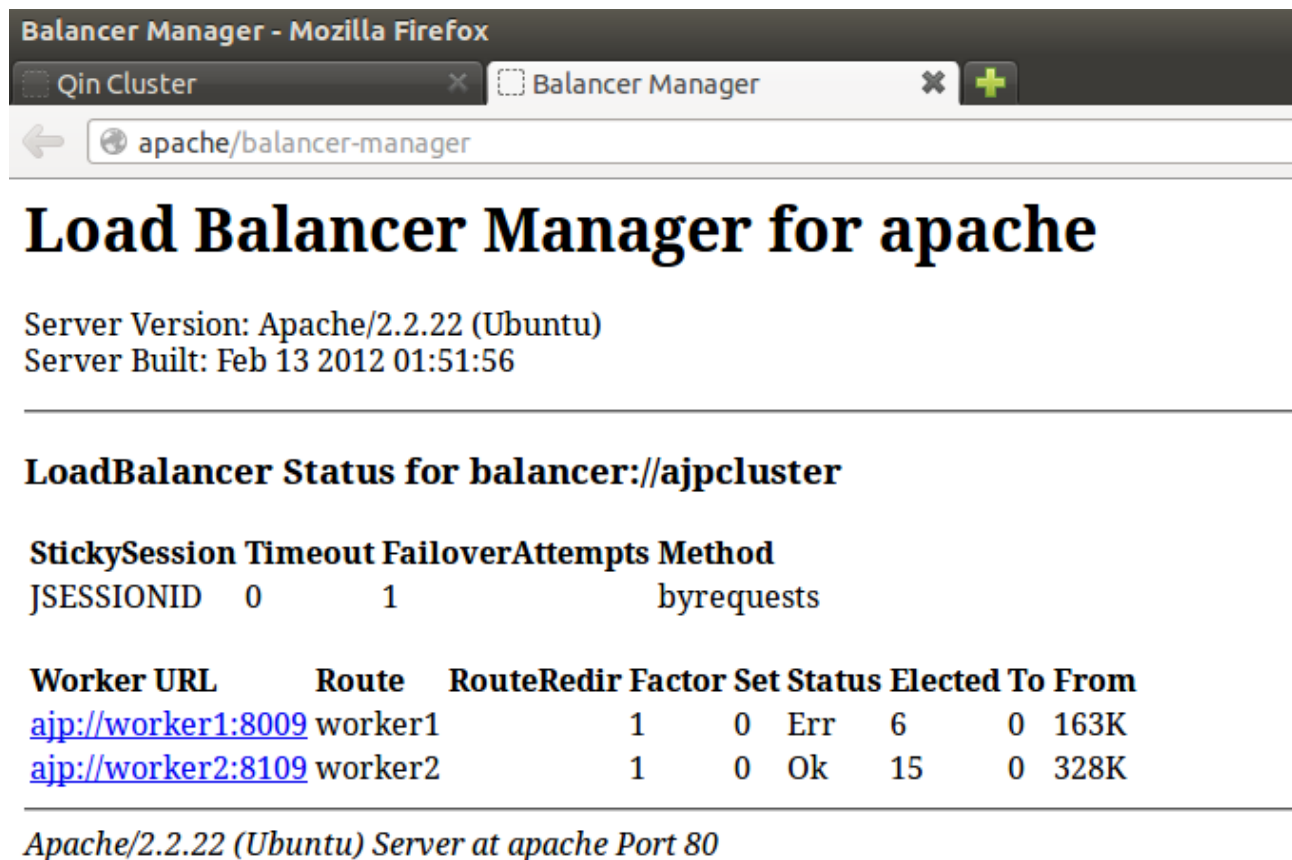
```
2012-10-17 04:36:05,251 INFO
```

```
[org.JBoss.ha.core.framework.server.DistributedReplicantManagerImp
```



```
l.qin] (AsyncViewChangeHandler Thread) I am (worker2:1199)
received membershipChanged event:
2012-10-17 04:36:05,252 INFO
[org.JBoss.ha.core.framework.server.DistributedReplicantManagerImp
l.qin] (AsyncViewChangeHandler Thread) Dead members: 1
([worker1:1099])
2012-10-17 04:36:05,252 INFO
[org.JBoss.ha.core.framework.server.DistributedReplicantManagerImp
l.qin] (AsyncViewChangeHandler Thread) New Members : 0 ([])
2012-10-17 04:36:05,252 INFO
[org.JBoss.ha.core.framework.server.DistributedReplicantManagerImp
l.qin] (AsyncViewChangeHandler Thread) All Members : 1
([worker2:1199])
2012-10-17 04:36:05,560 INFO
[org.JBoss.ha.core.framework.server.CoreGroupCommunicationService.
lifecycle] (Incoming-17,null) New cluster view for partition null
(id: 2, delta: -1, merge: false) : [worker2:1199]
2012-10-17 04:36:05,561 INFO
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(Incoming-17,null) Received new cluster view: [worker2:1199|2]
[worker2:1199]
(ajp-worker2%2F192.168.1.107-8109-1) PojoCacheManagerImpl:
existsKey
(ajp-worker2%2F192.168.1.107-8109-1) PojoCacheManagerImpl:
setValue
(ajp-worker2%2F192.168.1.107-8109-1) PojoCacheManagerImpl:
getValue
```

El balancer manager muestra que se perdió un nodo



Balancer Manager - Mozilla Firefox

Qin Cluster Balancer Manager

← apache/balancer-manager

Load Balancer Manager for apache

Server Version: Apache/2.2.22 (Ubuntu)
Server Built: Feb 13 2012 01:51:56

LoadBalancer Status for balancer://ajpcluster

StickySession	Timeout	FailoverAttempts	Method	Worker URL	Route	RouteRedir	Factor	Set	Status	Elected	To	From
JSESSIONID	0	1	byrequests	ajp://worker1:8009	worker1	1	0	Err	6	0	163K	
				ajp://worker2:8109	worker2	1	0	Ok	15	0	328K	

Apache/2.2.22 (Ubuntu) Server at apache Port 80

Ilustración 62: Balancer Manager mostrando que se cayó un nodo

- Se confirma en qué navegador de Internet se trabaja con qué usuario.



Ilustración 63: Recordando dónde están usuario1 y usuario2

Usuario2 en Chromium y usuario1 en Mozilla Firefox

- Se comprueba que ambos usuarios estén siendo atendidos por el nodo que no se bajó, en este caso worker2; se supera la prueba manual "Réplicación de sesión".

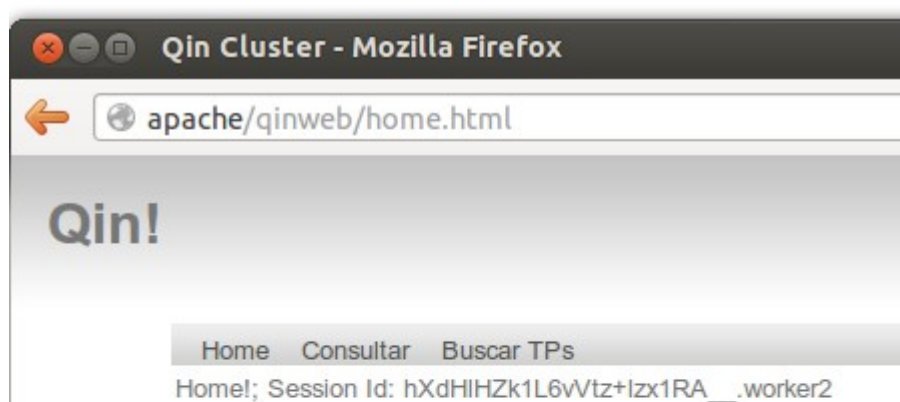
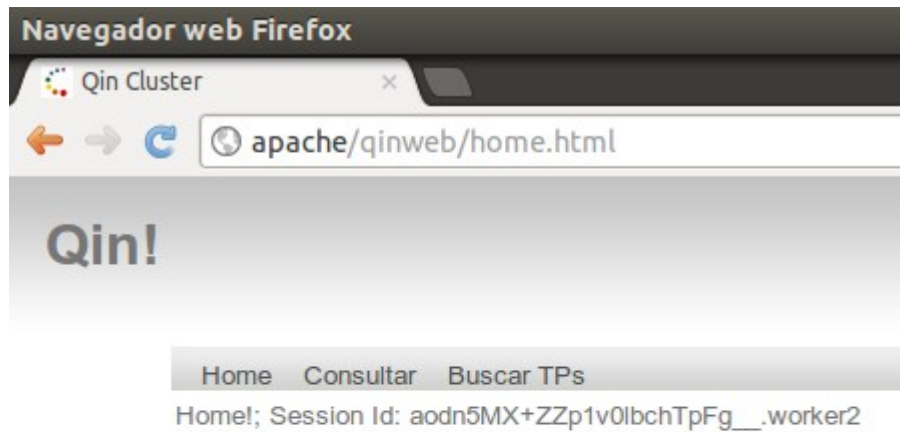


Ilustración 64: Ambos usuarios en worker2

- Se comprueba que el cache de segundo nivel de persistencia esté operando; se supera la prueba manual “*Caché*”.

MBean Inspector - Chromium

JBoss JMX Management Console

MBean Inspector

worker2:8180/jmx-console/HtmlAdaptor?action=inspectMBean&name=org.infinispan%3Atype%3DCache%2Cname%3Dcom.qin.entity.Material%28invalidation_sync%29%2Cr

Back to AgentRefresh MBean ViewWed Oct 17 04:37:34 ART 2012

Name

Domain name	org.infinispan
component	"com.qin.entity.Material(invalidation_sync)"
manager	Statistics
type	"hibernate"

Java Class

org.infinispan.interceptors.CacheMgmtInterceptor

Description

General statistics such as timings, hit/miss ratio, etc.

Attribute Name

Access

Type

Description

Attribute Value

Evictions

R

long

Number of cache eviction operations

0

RemoveMisses

R

long

Number of cache removals where keys were not found

0

ReadWriteRatio

R

double

read/writes ratio for the cache

2.5789473684210527

Dynamic MBean Description

R

java.lang.String

@MBean description

General statistics such as timings, hit/miss ratio, etc.

Hits

R

long

Number of cache attribute hits

49

NumberOfEntries

R

int

Number of entries currently in the cache

2

StatisticsEnabled

RW

boolean

Enables or disables the gathering of statistics by this component

☒ True☐ False

TimeSinceReset

R

long

Number of seconds since the cache statistics were last reset

715

ElapsedTime

R

long

Number of seconds since cache started

715

Misses

R

long

Number of cache attribute misses

0

Ilustración 65: Estadísticas de cache sobre Materia

- Se vuelve a levantar el nodo que se bajó anteriormente.

Worker1 retorna y vuelve a conectarse con worker2.

```
2012-10-17 04:57:20,702 INFO
[org.hornetq.core.server.cluster.impl.BridgeImpl] (Thread-1
(group:HornetQ-server-threads1424309400-1907685389)) Connecting
bridge sf.my-cluster.45ead1b5-1830-11e2-9578-68a3c49blef7 to its
destination [30969d9a-1830-11e2-b820-68a3c49blef7]
2012-10-17 04:57:21,146 INFO
[org.hornetq.core.server.cluster.impl.BridgeImpl] (Thread-1
(group:HornetQ-server-threads1424309400-1907685389)) Bridge sf.my-
cluster.45ead1b5-1830-11e2-9578-68a3c49blef7 is connected
[30969d9a-1830-11e2-b820-68a3c49blef7-> sf.my-cluster.45ead1b5-
1830-11e2-9578-68a3c49blef7]
2012-10-17 04:57:21,273 INFO
[org.JBoss.ha.framework.server.ClusterPartition.lifecycle.qin]
(Incoming-5,null) New cluster view for partition qin (id: 1,
delta: 1, merge: false) : [worker1:1099, worker2:1199]
2012-10-17 04:57:21,276 INFO
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(Incoming-5,null) Received new cluster view: [worker1:1099|1]
[worker1:1099, worker2:1199]
2012-10-17 04:57:21,276 INFO
[org.JBoss.ha.core.framework.server.DistributedReplicantManagerImp
l.qin] (AsynchViewChangeHandler Thread) I am (worker1:1099)
received membershipChanged event:
2012-10-17 04:57:21,276 INFO
[org.JBoss.ha.core.framework.server.DistributedReplicantManagerImp
l.qin] (AsynchViewChangeHandler Thread) Dead members: 0 ([])
2012-10-17 04:57:21,277 INFO
[org.JBoss.ha.core.framework.server.DistributedReplicantManagerImp
l.qin] (AsynchViewChangeHandler Thread) New Members : 1
([worker2:1199])
2012-10-17 04:57:21,277 INFO
[org.JBoss.ha.core.framework.server.DistributedReplicantManagerImp
l.qin] (AsynchViewChangeHandler Thread) All Members : 2
([worker1:1099, worker2:1199])
2012-10-17 04:57:28,732 INFO
[org.JBoss.ha.core.framework.server.CoreGroupCommunicationService.
lifecycle] (Incoming-12,null) New cluster view for partition null
(id: 1, delta: 1, merge: false) : [worker1:1099, worker2:1199]
2012-10-17 04:57:28,733 INFO
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(Incoming-12,null) Received new cluster view: [worker1:1099|1]
[worker1:1099, worker2:1199]
2012-10-17 04:57:30,244 INFO
[org.infinispan.remoting.transport.jgroups.JGroupsTransport]
(Incoming-5,null) Received new cluster view: [worker1:1099|1]
[worker1:1099, worker2:1199]
...
```

2012-10-17 05:01:00,315 WARN

```
[org.hornetq.core.server.cluster.impl.BridgeImpl] (Thread-5
(group:HornetQ-client-global-threads-1349107257)) sf.my-
cluster.30969d9a-1830-11e2-b820-68a3c49b1ef7::Connection failed
with failedOver=true: HornetQException[errorCode=4 message=The
connection was disconnected because of server shutdown]
    at
org.hornetq.core.client.impl.ClientSessionFactoryImpl$Channel0Hand
ler$1.run(ClientSessionFactoryImpl.java:1262) [:6.1.0.Final]
    at
org.hornetq.utils.OrderedExecutorFactory$OrderedExecutor$1.run(Ord
eredExecutorFactory.java:100) [:6.1.0.Final]
    at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecu
tor.java:1110) [:1.6.0_24]
    at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecu
tor.java:603) [:1.6.0_24]
    at java.lang.Thread.run(Thread.java:679) [:1.6.0_24]
```

2012-10-17 05:01:00,460 WARN

```
[org.hornetq.core.protocol.core.impl.ChannelImpl] (Old I/O client
worker (channelId: 236026373, /192.168.1.107:49462 =>
worker1/192.168.1.107:5445)) Can't find packet to clear: last
received command id 2 first stored command id 2
```

2012-10-17 05:01:00,464 WARN

```
[org.hornetq.core.protocol.core.impl.ChannelImpl] (Old I/O client
worker (channelId: 236026373, /192.168.1.107:49462 =>
worker1/192.168.1.107:5445)) Can't find packet to clear: last
received command id 3 first stored command id 3
```

2012-10-17 05:01:00,471 WARN

```
[org.hornetq.core.protocol.core.impl.ChannelImpl] (Old I/O client
worker (channelId: 236026373, /192.168.1.107:49462 =>
worker1/192.168.1.107:5445)) Can't find packet to clear: last
received command id 4 first stored command id 4
```

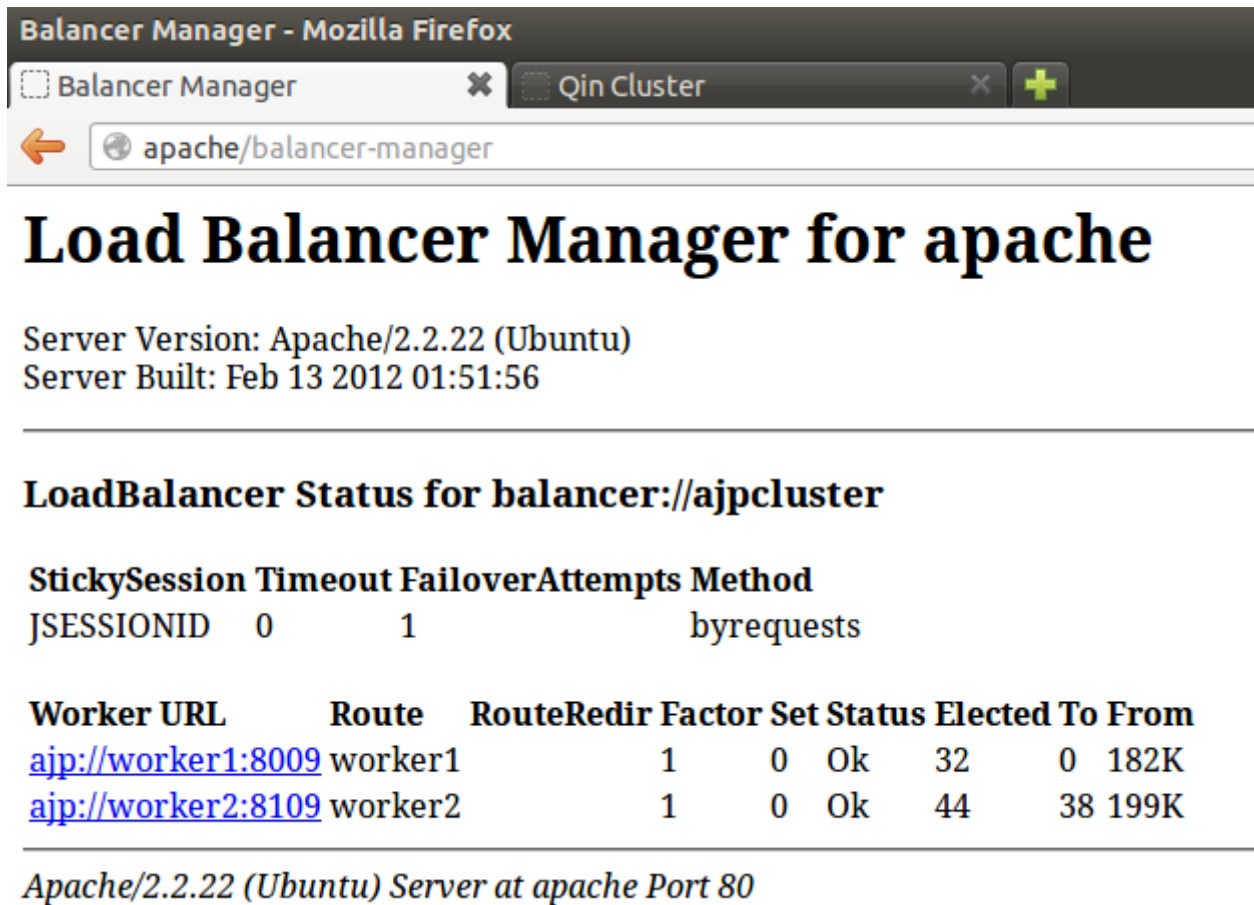
2012-10-17 05:01:00,483 WARN

```
[org.hornetq.core.protocol.core.impl.ChannelImpl] (Old I/O client
worker (channelId: 236026373, /192.168.1.107:49462 =>
worker1/192.168.1.107:5445)) Can't find packet to clear: last
received command id 5 first stored command id 5
```

2012-10-17 05:01:04,698 WARN

```
[org.hornetq.core.cluster.impl.DiscoveryGroupImpl] (hornetq-
discovery-group-thread-dg-group1) There are more than one servers
on the network broadcasting the same node id. You will see this
message exactly once (per node) if a node is restarted, in which
case it can be safely ignored. But if it is logged continuously it
means you really do have more than one node on the same network
active concurrently with the same node id. This could occur if you
have a backup node active at the same time as its live node.
nodeID=30969d9a-1830-11e2-b820-68a3c49b1ef7
```

- Se vuelve a armar el Cluster.



Balancer Manager for apache

Server Version: Apache/2.2.22 (Ubuntu)
Server Built: Feb 13 2012 01:51:56

LoadBalancer Status for balancer://ajpcluster

StickySession Timeout FailoverAttempts Method
JSESSIONID 0 1 byrequests

Worker URL	Route	RouteRedir	Factor	Set	Status	Elected	To	From
ajp://worker1:8009	worker1		1	0	Ok	32	0	182K
ajp://worker2:8109	worker2		1	0	Ok	44	38	199K

Apache/2.2.22 (Ubuntu) Server at apache Port 80

Ilustración 66: Cluster rearmado

Como se puede apreciar, las pruebas manuales que sirvieron para regular el desarrollo del trabajo práctico, son igualmente superadas por este Cluster en esta etapa de producto terminado, lo cual es de vital importancia porque cada prueba manual encierra propiedades que son a nuestro criterio deseables en una estructura de Cluster de un ambiente productivo, y además cada prueba manual superada revalida la investigación efectuada.

Terracotta + Tomcat

En el caso de Terracotta + Tomcat, se provee a continuación evidencia de una corrida de las características que permiten comprobar las pruebas manuales.

- Se levanta el Cluster; se comprueba que se haya armado correctamente.

Aplican a este caso, las mismas imágenes de la aplicación de prueba qinweb y del balancer manager como en el caso de JBoss.

Verificación de que se ven ambos nodos (con dos conexiones cada uno) en la consola adminitrativa de Terracotta.

The screenshot displays the Terracotta Developer Console interface. The top status bar shows system information: 10°C, battery level, and the user 'diego' logged in on October 17, 2024, at 05:24. The main window is divided into several sections:

- Left Sidebar:** Contains navigation links for File, Tools, and Help, along with icons for Ehcache, Sessions, Quartz, and Hibernate.
- Top Table:** A table listing cluster nodes with columns for Host, Port, Client ID, and Live Object Count.

Host	Port	Client ID	Live Object Count
localhost		59233	5
localhost		59171	2
localhost		59164	1
localhost		59229	4
			157
- Main View:** A large area showing a hierarchical tree of the cluster's internal components, including 'Terracotta cluster', 'My application', 'Ehcache', 'Sessions', 'Monitoring', 'Runtime statistics', 'Garbage collection', 'Topology', 'Connected clients (4)', 'Server Array (1)', and 'Platform'. The 'Connected clients' section is expanded, showing four client IDs: ClientID[1] localhost:59164, ClientID[2] localhost:59171, ClientID[4] localhost:59229, and ClientID[5] localhost:59233.
- Bottom Status Bar:** A message indicating 'Added new DSO client: localhost:59233'.

Ilustración 67: Consola de Terracotta

- Se ingresa con el navegador Mozilla Firefox con usuario1.

Misma imagen que en el caso de JBoss.

- Se confirma qué nodo atiende a usuario1; es el worker2.

Misma imagen que en el caso de JBoss.

- Se ingresa con el navegador Google Chrome con usuario2.

Misma imagen que en el caso de JBoss.

- Se confirma qué nodo atiende a usuario2; es el worker1.

Misma imagen que en el caso de JBoss.

- Se verifica que el sincronizador está trabajando.

El sincronizador es Terracotta, por ende sí está trabajando, ya que estamos usando su consola administrativa.

- Se baja worker1; comprobar que la sincronización continúa; se superan las pruebas manuales “Desconexión de un nodo en esquema colaborativo” y “Disponibilidad del sistema”.

Aplica la imagen del balancer manager mostrando que se cayó un nodo vista en el caso de JBoss.

Confirmación en la consola administrativa de Terracotta, de que faltan dos conexiones.

The screenshot displays the Terracotta Developer Console interface. The top status bar shows the user 'diego' and the time 'mié 17 de oct 05:23'. The main window is divided into several sections. On the left, there is a sidebar with icons for 'Sessions', 'Quartz', 'Hibernate', and 'Ehcache'. The 'Sessions' section is currently active, showing a table with columns: Host, Port, Client ID, Live Object Count, and Live Object Count. The table contains two rows of data:

Host	Port	Client ID	Live Object Count	Live Object Count
localhost		59171	2	2,602
localhost		59164	1	411

Below the table, there is a diagram of the cluster topology. It shows a 'Terracotta cluster' with nodes 'My application', 'Ehcache', 'Hibernate', 'Sessions', 'Monitoring', 'Runtime statistics', 'Garbage collection', and 'Topology'. The 'Topology' section is expanded, showing 'Connected clients (2)' with details for 'ClientID[1] localhost:59164' and 'ClientID[2] localhost:59171'. A 'Server Array (1)' is also shown. The bottom status bar indicates 'Detached DSO client: localhost:59161'.

Ilustración 68: Consola de Terracotta con nodo caído

- Se confirma en qué navegador de Internet se trabaja con qué usuario.

Misma imagen que en el caso de JBoss.

- Se comprueba que ambos usuarios estén siendo atendidos por el nodo que no se bajó, en este caso worker2; se supera la prueba manual *"Réplicación de sesión"*.

Misma imagen que en el caso de JBoss.

- Se comprueba que el cache de segundo nivel de persistencia esté operando; se supera la prueba manual "Caché".

Estadísticas de ehcache.

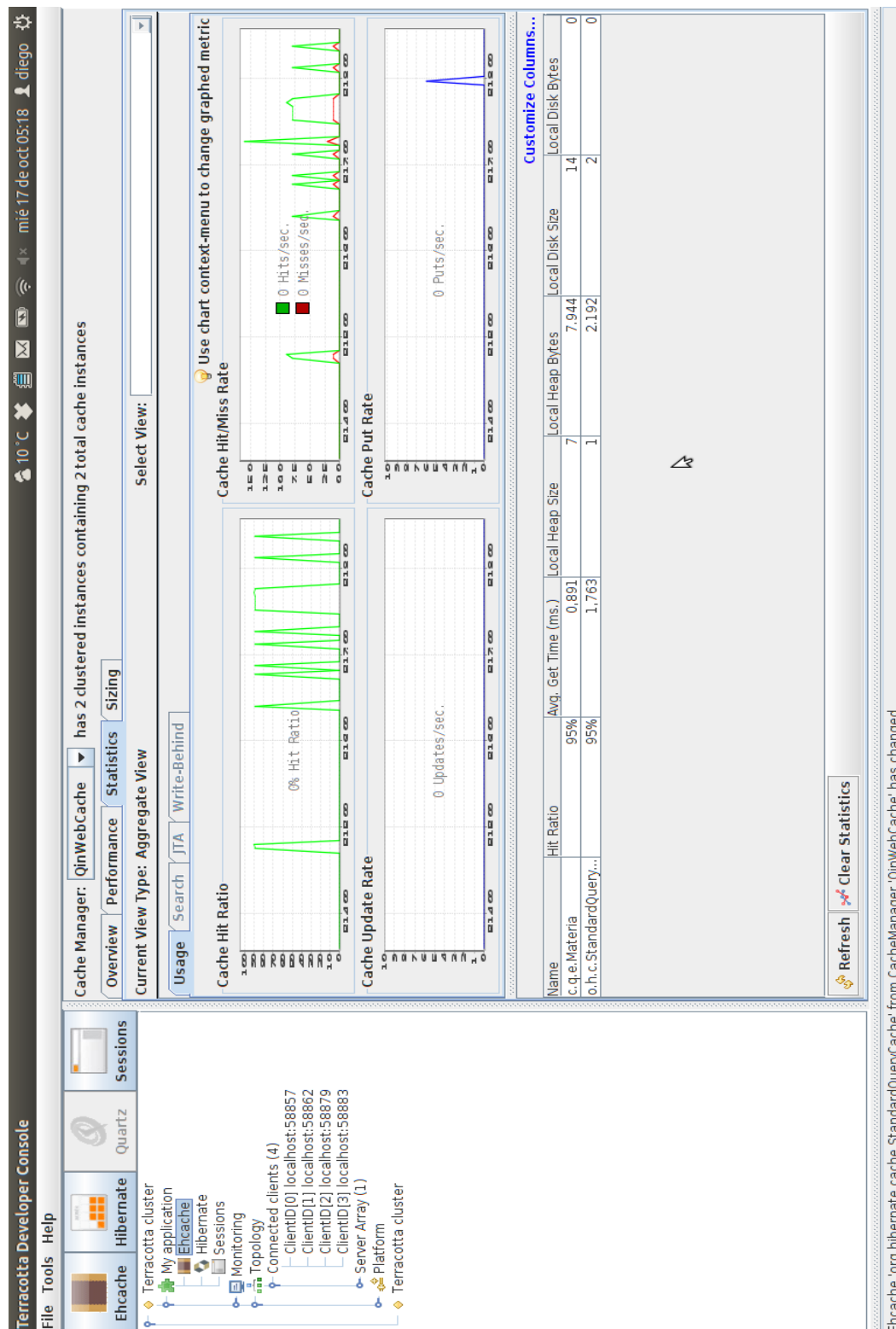


Ilustración 69: Ehcache

Estadísticas de hibernate.

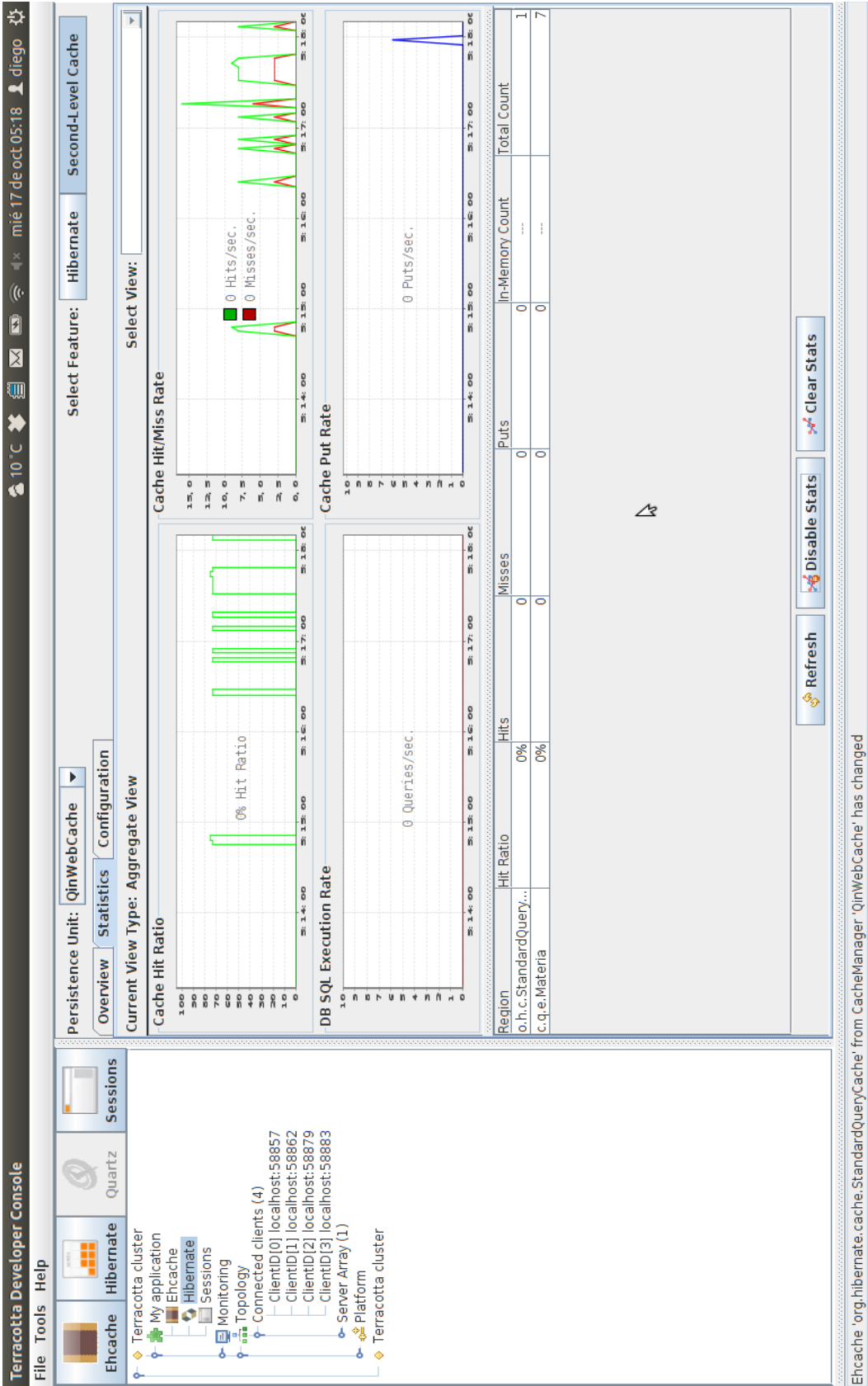


Ilustración 70: Hibernate

- Se vuelve a levantar el nodo que se bajó anteriormente.

Se confirma que se vuelven a ver ambos nodos (dos conexiones por nodo).

The screenshot shows the Terracotta Developer Console interface. The top bar includes system status (10°C, battery, network) and the user 'diego'. The main window is divided into two panes. The left pane shows a tree view of the cluster structure, including 'Terracotta cluster', 'My application', 'Ehcache', 'Hibernate', 'Sessions', 'Runtime statistics', 'Garbage collection', 'Topology', 'Connected clients (4)', 'ClientID[1] localhost:59164', 'ClientID[2] localhost:59171', 'ClientID[6] localhost:59255', 'ClientID[7] localhost:59259', 'Server Array (1)', 'Platform', and 'Terracotta cluster'. The right pane shows a table of client connections.

Host	Port	Client ID	Live Object Count
localhost		59171	2
localhost		59259	7
localhost		59164	1
localhost		59255	6
			0

At the bottom right, a status bar indicates 'Added new DSO client: localhost:59259'.

Ilustración 71: Consola de Terracotta con todos los nodos

También en este caso, se superan las pruebas manuales, al igual que en el caso de JBoss. Aplica la misma conclusión que en el caso de JBoss.

8. Datos de los autores

Barrabino, Diego

Datos personales:

- Apellido y nombre: BARRABINO, Diego.
- Documento Nacional de Identidad (D.N.I.): 29.064.455.
- Fecha de nacimiento: 7 de septiembre de 1981.
- Edad: 31 años.
- Domicilio: Quito 4247 1º 3, CPA C1212ABM, entre Mármol y Muñiz, Almagro, Ciudad Autónoma de Buenos Aires.
- Teléfono particular: 011 – 3528 – 6888.
- Casilla de e-mail: dbarrabino@gmail.com.

Antecedentes laborales:

a).- Certant Technology Solutions (Av. De Mayo 666, 4º piso – Capital Federal. Teléfono: 5219-0855/6).

Pasantía de seis meses (diciembre 2003 – junio 2004) en la citada empresa de informática. Se realizaron las siguientes actividades, en las oficinas de la mencionada empresa tanto como en las instalaciones del cliente: programación, mantenimiento, análisis de proyectos, análisis de programas y documentación de proyectos.

Durante los últimos tres meses de los seis citados, presté servicios en UOL – Sinectis, en el departamento de Sistemas, área de Tecnología, sita en Florida 537 6º piso (1005), donde se efectuaron tareas de mantenimiento variadas, programación en Perl, tanto para desarrollar nuevos programas o scripts como para mantener y supervisar programas o scripts anteriores o de otros programadores, HTML y consultas básicas a bases de datos en SQL.

b).- Dirección de Informática – Gerencia de Organización y Sistemas- CONICET (Av. Rivadavia 1906, 2º piso, oficina 13 – Capital Federal. Teléfono: 5983 – 1420, internos 601 y 509).

Pasantía desde el primero de octubre de 2004 hasta abril de 2005, bajo contrato desde entonces.

Conformo un grupo de trabajo de aproximadamente 30 personas directamente bajo supervisión del director de Informática del CONICET, cuyo objetivo es desarrollar, implementar, relevar, mantener y proponer aplicaciones informáticas de diverso porte y amplia funcionalidad según el caso, siempre alineadas con los objetivos estratégicos del CONICET. Las herramientas utilizadas son Java, Eclipse, EnterpriseArchitect, MySQL Workbench, Jira, Tomcat, Glassfish, Hibernate, EclipseLink, Struts1 y 2, JSP, JavaScript, MySQL, Oracle, HTML, etc. Actualmente los proyectos más notorios son el sistema SIGEVA (Sistema Integral de Gestión y Evaluación), la implantación como proyecto del sistema SIGEVA en otras instituciones, el sistema DFD que provee de Firma Digital (implementación completa; primer caso en el Estado Argentino) al sistema SIGEVA, el sistema SIGERH (Sistema Integral de Gestión de Recursos Humanos), entre otros, como ser el sistema CER de Certificaciones de Servicio, el sistema MEC de Mesa de Entrada del CONICET, etc. Con el correr del tiempo, he participado incluso desde sus orígenes en todos los proyectos.

En el grupo mencionado, ejecuto tareas de relevamiento, análisis, diseño, desarrollo, implementación, control y seguimiento de diversas funcionalidades de los sistemas administrados, en un marco de colaboración total y recíproca con absolutamente todos mis compañeros, varios de los cuales conformamos un núcleo estable desde hace varios años.

Antecedentes académicos:

CBC (año 2000)

Álgebra (código 27). Curso 42708. Nota: 7. Primer cuatrimestre de 2000.

Introducción al Pensamiento Científico (código 40). Curso 44001. Nota: 9. Primer cuatrimestre de 2000.

Física (código 03). Curso 40302. Nota: 7. Primer cuatrimestre de 2000.

Análisis Matemático 1 (código 28). Curso 42808. Nota: 8. Segundo cuatrimestre de 2000.

Sociedad y Estado (código 24). Curso 42422. Nota: 8. Segundo cuatrimestre de 2000.

Química (código 05). Curso 40504. Nota: 9. Segundo cuatrimestre de 2000.

Carrera 10 – Ingeniería en Informática (año 2001 a la fecha – padrón 80183)

Algoritmos y Programación I (programación en Pascal) (código 7540). Cátedra: López - Vega. Nota: 9 (11/07/2001 L85 F43). Primer cuatrimestre de 2001.

Física I A (código 6201). Cátedra: Menikheim. Nota: 7 (31/07/2001 L100 F210). Final revisado por la prof. Lombardo. Primer cuatrimestre de 2001.

Análisis Matemático II A (código 6103). Cátedra: Patetta. Nota: 10 (11/12/2001 L148 F52). Segundo cuatrimestre de 2001.

Química (código 6301). Cátedra: Kim – Roble. Nota: 8 (19/12/2001 L68 F217). Segundo cuatrimestre de 2001.

Álgebra II A (código 6108). Cátedra: Álvarez Juliá. Nota: 6 (27/12/2001 L146 F143). Segundo cuatrimestre de 2001.

Algoritmos y Programación II (programación en Pascal) (código 7541). Cátedra: Mandrafina. Nota: 7 (06/03/2002 L86 F133). Segundo cuatrimestre de 2001.

Algoritmos y Programación III (programación en Delphi) (código 7507). Cátedra: Fíntela. Nota: 9 (16/07/2002 L87 F2). Primer cuatrimestre de 2002.

Análisis Matemático III A (código 6110). Cátedra: Murmis. Nota: 5 (19/07/2002 L143 F212). Primer cuatrimestre de 2002.

Probabilidad y Estadística B (código 6109). Cátedra: Busch. Nota: 4 (19/12/2002 L147 F124). Final revisado por el prof. Sacerdotti. Segundo cuatrimestre de 2002.

Análisis Numérico I (código 7512). Cátedra: Tarela - Cavaliere. Nota: 7 (16/12/2002 L87 F205). Segundo cuatrimestre de 2002.

Física II A (código 6203). Cátedra: Cremaschi – Mesaros – Hogart. Nota: 7 (18/02/2003 L102 F33). Final revisado por el prof. Leone. Segundo cuatrimestre de 2002.

Laboratorio (código 6602). Cátedra: Bertuccio – Reiser. Nota: 5 (18/07/2003 L128 F214). Primer cuatrimestre de 2003.

Física III D (código 6215). Cátedra: Arcondo. Nota: 8 (08/07/2003 L102 F100). Primer cuatrimestre de 2003.

Estructura del Computador (código 6670). Cátedra: M. C. Ginzburg. Nota: 9 (15/08/2003 L129 F51). Primer cuatrimestre del 2003.

Simulación (programación en GPSS; código 7526). Cátedra: Nota: 4 (17/12/2003 L89 F175). Segundo cuatrimestre de 2003.

Organización de Computadoras (código 6620). Cátedra: Hamkalo. Nota: 10 (11/12/2003 L129 F96). Segundo cuatrimestre de 2003.

Taller de Programación I (código 7542). Cátedra: Veiga. Nota: 8 (26/07/2005 L92 F238). Primer cuatrimestre de 2005.

Organización de Datos (código 7506). Cátedra: Saubidet. Nota: 7 (11/12/2006 L95 F104). Segundo cuatrimestre de 2006.

Sistemas Operativos (código 7508). Cátedra: Clúa. Nota 10 (05/07/2007 L96 F146). Primer cuatrimestre de 2007.

Análisis de la información (código 7509). Cátedra: González. Nota 5 (10/07/2007 L96 F163). Primer cuatrimestre de 2007.

Estructura de las Organizaciones (código 7112). Cátedra: Barmack. Nota: 8 (27/12/2007 L145 F207). Segundo cuatrimestre de 2007.

Modelos y Optimización I (código 7114). Cátedra: Ramonet. Nota: 8 (26/12/2007 L145 F199). Segundo cuatrimestre de 2007.

Técnicas de Diseño (código 7510). Cátedra: Pantaleo. Nota: 8 (07/07/2008 L98 F186). Primer cuatrimestre de 2008.

Información en las Organizaciones (código 7113). Cátedra: Zambrano. Nota: 6 (10/07/2008 L146 F149). Primer cuatrimestre de 2008.

Base de Datos (código 7515). Cátedra: Alé. Nota: 5 (07/10/2009 L102 F84). Segundo cuatrimestre de 2008.

Introducción a los Sistemas Distribuidos (código 7543). Cátedra: Hamelin. Nota: 5 (23/02/2010 L103 F82). Segundo cuatrimestre de 2008.

Taller de Programación II (código 7552). Cátedra: Servetto. Nota: 10 (05/03/2010 L103 F176). Primer cuatrimestre de 2009.

Administración y Control de Proyectos Informáticos I (código 7544). Cátedra: Martínez. Nota: 7 (28/09/2009 L102 F45). Primer cuatrimestre de 2009.

Taller de Desarrollo de Proyectos I (código 7545). Cátedra: Pineiro – Pignataro. Nota: 9 (15/02/2010 L103 F31). Segundo cuatrimestre de 2009.

Legislación y Ejercicio Profesional de la Ingeniería en Informática (código 7140). Cátedra: Papaleo – Noremberg. Nota: 7 (22/12/2009 L149 F120). Segundo cuatrimestre de 2009.

Taller de Desarrollo de Proyectos II (código 7547). Cátedra: Fontela. Nota: 9 (06/07/2010 L104 F13). Primer cuatrimestre de 2010.

Administración y Control de Proyectos Informáticos II (código 7546). Cátedra: Martínez. Nota: 6 (27/07/2010 L104 F153). Primer cuatrimestre de 2010.

Introducción a los Sistemas Inteligentes (código 7550). Cátedra: Ochoa. Nota: 10 (13/12/2010 L105 F96). Segundo cuatrimestre de 2010.

Calidad en el Desarrollo de Sistemas (código 7548). Cátedra: Pantaleo. Nota: 5 (15/12/2010 L105 F119). Segundo cuatrimestre de 2010.

Idioma Inglés (código 7801). Cátedra: Johnstone. Nota: 10 (28/03/2011 L023 F81). Primer cuatrimestre de 2011.

Manufactura Integrada por Computadora (CIM) I (código 7565). Cátedra: Ierache – Ochoa. Nota: 7,5 (30/05/2011 L106 F133). Primer cuatrimestre de 2011.

Sistemas Automáticos de Diagnóstico y Detección de Fallas I (código 7567). Cátedra: Merlino. Nota: 8 (01/07/2011 L106 F135). Primer cuatrimestre de 2011.

Sistemas Automáticos de Diagnóstico y Detección Fallas II (código 7569). Cátedra: Merlino. Nota: 9 (02/12/2011 L107 F203). Segundo cuatrimestre de 2011.

Manufactura Integrada por Computadora (CIM) II (código 7566). Cátedra: Ierache – Ochoa. Nota: 9 (05/12/2011 L107 F210). Segundo cuatrimestre de 2011.

Criptografía y Seguridad Informática (código 6669). Cátedra: Pagola – Wald – Caracoche. Nota: 7 (19/12/2011 L41 F120). Segundo cuatrimestre de 2011.

Estadísticas a la fecha:

- CBC: 6 materias, promedio 8.
- Carrera 10 – Ingeniería en Informática: 40 materias, promedio 7,46.
- Suma de créditos: 236.

Moreyra Martín

Datos personales:

- Apellido y nombre: Moreyra, Martín Jorge.
- Documento Nacional de Identidad (D.N.I.): 30.892.909.
- Fecha de nacimiento: 6 de junio de 1984.
- Edad: 28 años.
- Domicilio: Honduras 4564 – 3/B, CP 1414, Ciudad Autónoma de Buenos Aires.
- Teléfono particular: 011 – 4195 – 3792.
- Casilla de e-mail: moreyramj@gmail.com.

Antecedentes laborales:

a) Globant- (Febrero 2006 – Junio 2009):

Participación en diferentes proyectos como desarrollador: red social (FunkySexyCool), sistema de búsqueda y compra de vuelos (Orbitz), sistema de administración de información sobre vuelos (OAG). Herramientas utilizadas: Java, JSP, Velocoty, Struts, Spring, Hibernate, MySQL, Tomcat.

b) ITR- (Julio 2009 - Diciembre 2009):

Proyecto para Swiss Medical, Parte del grupo de desarrollo del sistema que maneja historiales médicos, internaciones y otros. Las herramientas utilizadas fueron: J2EE, Servlets, Sybase portal y Sybase DB.

c) Swiss Medical - (Enero 2010 – Septiembre 2010):

Ídem b) pero en relación de dependencia directa con Swiss Medical.

d) OSPIM - (Agosto 2010 – Presente):

Formo parte de un grupo de desarrolladores quienes estamos trabajando sobre el nuevo sistema interno de la obra social. El sistema comprende diferentes áreas, afiliaciones, contabilidad, tesorería, entre otros. Mis tareas comprenden desde el relevamiento de los requerimientos, su documentación, diseño y desarrollo. Las herramientas utilizadas son: Java, Servlets, Struts, Liferay, JSP, Postgres, Tomcat.

Antecedentes académicos:

CBC (año 2003)

Álgebra (código 27)

Introducción al Pensamiento Científico (código 40). Curso 44001.

Física (código 03). Curso 40302.

Análisis Matemático 1 (código 28). Curso 42808.

Sociedad y Estado (código 24). Curso 42422.

Química (código 05). Curso 40504.

Carrera 10 – Ingeniería en Informática (año 2004 a la fecha – padrón 84394).

Física I A (código 6201) Nota: 6 (10/08/2004 L103 F127).

Algoritmos Y Programacion I (código 7540) Nota: 6 (13/08/2004 L91 F109).

Algebra II A (código 6108) Nota: 5 (09/12/2004 L150 F65).

Algoritmos Y Programacion II (código 7541) Nota: 9 (17/12/2004 L91 F188).

Analisis Matematico II A (código 6103) Nota: 4 (22/02/2005 L151 F16).

Quimica (código 6301) Nota: 6 (14/07/2005 L71 F218).

Algoritmos Y Programacion III (código 7507) Nota: 7 (08/08/2005 L93 F37).

Analisis Numerico I (código 7512) Nota: 7 (12/08/2005 L93 F60).

Física II A (código 6203) Nota: 5 (15/12/2005 L104 F98).

Probabilidad Y Estadística B (código 6109) Nota: 5 (03/03/2006 L152 F59).

Estructura Del Computador (código 6670) Nota: 8 (13/07/2006 L132 F207).

Laboratorio (código 6602) Nota: 6 (13/07/2006 L132 F217).

Física III D (código 6215) Nota: 6 (R 08/08/2006 L105 F33).

Organizacion De Datos (código 7506) Nota: 8 (19/07/2007 L96 F206).

Sistemas Operativos (código 7508) Nota: 10 (27/12/2007 L97 F202).

Analisis Matematico III A (código 6110) Nota: 6 (25/02/2008 L149 F124).

Taller De Programacion I (código 7542) Nota: 9 (08/07/2008 L98 F202).

Estructura De Las Organizaciones (código 7112) Nota: 4 (30/07/2008 L147 F5).

Organización De Computadoras (código 6620) Nota: 6 (14/08/2008 L136 F79).

Analisis De La Informacion (código 7509) Nota: 6 (15/12/2008 L100 F11).

Modelos Y Optimizacion I (código 7114) Nota: 8 (16/02/2009 L147 F230).

Informacion En Las Organizaciones (código 7113) Nota: 4 (24/02/2009 L148 F31).

Simulación (código 7526) Nota: 10 (05/08/2009 L101 F106).

Técnicas De Diseño (código 7510) Nota: 7 (10/08/2009 L101 F133).

Taller De Programación II (Código 7552) Nota: 10 (21/08/2009 L101 F219).

Taller De Desarrollo De Proyectos I (Código 7545) Nota: 9 (08/02/2010 L102 F238).

Introducción A Los Sistemas Inteligentes (Código 7550) Nota: 6 (22/02/2010 L103 F88).

Introducción A Los Sistemas Distribuidos (Código 7543) Nota: 7 (23/02/2010 L103 F82).

Base De Datos (Código 7515) Nota: 7 (21/07/2010 L104 F114).

Legislación y Ejercicio Profesional de la Ingeniería en Informática (código 7140). Nota: 5. Cátedra: Papaleo – Noremburg. (22/12/2009 L152 F230).

Administración y Control de Proyectos Informáticos I (código 7544). Nota: 4. Cátedra: Martínez. (10/08/2011 L107 F141).

Calidad en el Desarrollo de Sistemas (código 7548). Nota: 7 Cátedra: Pantaleo (12/12/2011 L107 F245).

Manufactura Integrada por Computador (CIM) I (código 7565). Nota: 7 Cátedra: Ierache - Ochoa (30/06/2011 L106 F166).

Sistemas Automáticos de Diagnóstico y Detección de Fallas I (código 7567). Nota: 8. Cátedra: Merlino (27/06/2011 L106 F135).

Taller de Desarrollo de Proyectos II (código 7547). Nota: 8. Cátedra: Fontela (3/07/2012 L109 F52).

Manufactura Integrada por Computadora (CIM) II (código 7566). Nota: 9. Cátedra: Ierache - Ochoa (5/12/2011 L107 F210).

Sistemas Automáticos de Diagnóstico y Detección de Fallas II (código 7569) Nota: 9 (02/12/2011 L107 F203).

Criptografía y seguridad informática (código 6669) Nota: 8 (19/12/2011 L141 F120).

Administración y Control de Proyectos Informáticos II (código 7546). Nota: 4 Cátedra: Martínez (31/07/2012 L109 F197).

Arquitectura de Software (código 7573) Nota: 7 (05/07/2012 L109 F69).

Estadísticas a la fecha:

- CBC: 6 materias.
- Carrera 10 – Ingeniería en Informática: 40 materias, Promedio 6,83.
- Suma de créditos: 236.

9. Bibliografía

[CLOUD BEST PRACTICES] VARIA, Jinesh. *Architecting for the Cloud: Best Practices* [en línea]. Amazon, enero de 2010. Disponible en versión PDF en Internet:

<http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf>

[CLUSTER] Wikipedia. *Cluster (Informática)* [en línea]. Wikipedia, 2012 [ref. de julio 2011 a julio de 2012]. Disponible en Internet:

<http://es.wikipedia.org/wiki/Cluster_%28inform%C3%A1tica%29>

[CLUSTERING GUIDE 5.1] STANSBERRY, Brian. ZAMARRENO, Galder.

FERRARO, Paul Ferraro. *JBoss AS 5.1 Clustering Guide - High Availability Enterprise Services with JBoss Application Server Clusters* [en línea].

Comunidad JBoss. KITTOLI, Samson, septiembre de 2009 [ref. de julio 2011 a julio de 2012]. Disponible en Internet:

<http://docs.jboss.org/jbossclustering/cluster_guide/5.1/html-single/index.html>. Igualmente disponible en versión PDF en Internet:

<http://docs.jboss.org/jbossclustering/cluster_guide/5.1/pdf/Clustering_Guide.pdf>

[CLUSTERING GUIDE 5.1 CH 4] _____. *JBoss AS 5.1 Clustering Guide - High Availability Enterprise Services with JBoss Application Server Clusters* [en línea]. Comunidad JBoss. KITTOLI, Samson, septiembre de 2009 [ref. de julio 2011 a julio de 2012]. Capítulo 4. Clustered Deployment Options. Disponible en Internet: <http://docs.jboss.org/jbossclustering/cluster_guide/5.1/html-single/index.html#deployment.chapt>

[CLUSTERED JAVA EE] _____. *JBoss AS 5.1 Clustering Guide - High Availability Enterprise Services with JBoss Application Server Clusters* [en línea]. Comunidad JBoss. KITTOLI, Samson, septiembre de 2009 [ref. de julio 2011 a julio de 2012]. Parte II. Clustered Java EE. Disponible en Internet:

<http://docs.jboss.org/jbossclustering/cluster_guide/5.1/html-single/index.html#Clustered_JEE>

[CLUSTERED JAVA EE - START] _____. *JBoss AS 5.1 Clustering Guide - High Availability Enterprise Services with JBoss Application Server Clusters* [en línea]. Comunidad JBoss. KITTOLI, Samson, septiembre de 2009 [ref. de julio 2011 a julio de 2012]. Capítulo I. Introduction and Quick Start. Disponible en Internet: <http://docs.jboss.org/jbossclustering/cluster_guide/5.1/html-single/index.html#clustering-intro.chapt>

[EJB3 IN ACTION] PANDA, Debu. RAHMAN, Reza. LANE, Derek. *EJB3 In Action*. Manning, 2007. ISBN 1-933988-34-7. Segunda edición. 677 p.

[FIELDING] FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures* [en línea]. Universidad de California [Irvine, California, Estados Unidos]. Comité de disertación (Profesor Richard N. Taylor, Chair; Profesor Mark S. Ackerman; Profesor David S. Rosenblum), 2000 [ref. de julio 2011 a julio de 2012]. Disponible en Internet: <<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>>. Igualmente disponible en versión PDF en Internet: <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>

[INFINISPAN] ZAMARREÑO, Galder. *Using Infinispan as JPA-Hibernate Second Level Cache Provider* [en línea]. Comunidad JBoss. Grinovero, Sanne, 18/02/2012 [ref. de julio 2011 a julio de 2012]. Disponible en Internet: <<https://docs.jboss.org/author/display/ISPN/Using+Infinispan+as+JPA-Hibernate+Second+Level+Cache+Provider>>

[JBoss AS IN ACTION] JAMAE, Javid. JOHNSON, Peter. *JBoss In Action - Installation, configuration, and deployment*. Manning, 2009. ISBN 978-1-933988-02-3. 464 p.

[JBoss AS INSTALLATION] JBoss Community. *JBoss Application Server - Installation And Getting Started Guide* [en línea]. 2008 [ref. de julio 2011 a julio de 2012]. Disponible en Internet: <http://docs.jboss.org/jbossas/docs/Installation_And_Getting_Started_Guide/5/html_single/index.html>. Igualmente disponible en versión PDF en Internet: <http://docs.jboss.org/jbossas/docs/Installation_And_Getting_Started_Guide/5/pdf/Installation_And_Getting_Started_Guide.pdf>

[PEAA] FOWLER, Martin. *Patterns of Enterprise Application Architecture*. RICE, Dave. FOEMMEL, Matthew. HIEATT, Edward. MEE, Robert. STAFFORD, Randy. Addison - Wesley, 2003. The Addison - Wesley Signature Series. ISBN 0-321-12742-0

[POJO FAQ] Wang, Ben. Marlow, Scott. Greene, Jason. *Frequently Asked Questions About Pojo Cache* [en línea]. Release 3.0.0, noviembre de 2008 [ref. de julio 2011 a julio de 2012]. Disponible en Internet: <http://www.jboss.org/file-access/default/members/jboss-cache/freezone/docs/3.0.0.GA/pojo/faq_en/html_single/index.html>. Igualmente disponible en versión PDF en Internet: <http://www.jboss.org/file-access/default/members/jboss-cache/freezone/docs/3.0.0.GA/pojo/faq_en/pdf/faq_en.pdf>

[POJO TUTORIAL] Wang, Ben. Zamarreño, Galder. *PojoCache Tutorial* [en línea]. Release 3.0.0, noviembre de 2008 [ref. de julio 2011 a julio de 2012]. Disponible en Internet: <http://www.jboss.org/file-access/default/members/jboss-cache/freezone/docs/3.0.0.GA/pojo/tutorial_en/html_single/index.html>. Igualmente disponible en versión PDF en Internet: <http://www.jboss.org/file-access/default/members/jboss-cache/freezone/docs/3.0.0.GA/pojo/tutorial_en/pdf/tutorial_en.pdf>

[POJO USER GUIDE] Wang, Ben. Greene, Jason. *PojoCache User Documentation* [en línea]. Release 3.0.0, noviembre de 2008 [ref. de julio 2011 a julio de 2012]. Disponible en Internet: <http://www.jboss.org/file-access/default/members/jboss-cache/freezone/docs/3.0.0.GA/pojo/userguide_en/html_single/index.html>. Igualmente disponible en versión PDF en Internet: <http://www.jboss.org/file-access/default/members/jboss-cache/freezone/docs/3.0.0.GA/pojo/userguide_en/pdf/userguide_en.pdf>

[RESPONSIVENESS] Sun Microsystems Inc. *Java(TM) Look and Feel Design Guidelines: Advanced Topics*. Addison-Wesley, 2001. ISBN 978-0201775822. 200 p.

[SPRING] JOHNSON, Rod. HOELLER, Juergen. ARENDSSEN, Alef. SAMPALEANU, Colin. HARROP, Rob. RISBERG, Thomas. DAVISON, Darren. KOPYLENKO, Dmitriy. POLLACK, Mark. TEMPLIER, Thierry. VERVAET, Erwin. TUNG, Portia. HALE, Ben. COLYER, Adrian. LEWIS, John. LEAU, Costin. EVANS, Rick. *The Spring Framework - Reference Documentation* [en línea]. Comunidad SpringSource. [ref. de julio 2011 a julio de 2012]. Disponible en Internet: <<http://static.springsource.org/spring/docs/2.0.x/reference>>.

[SPRING MVC] _____. *The Spring Framework - Reference Documentation* [en línea]. Comunidad SpringSource. [ref. de julio 2011 a julio de 2012]. Capítulo 13 Web MVC framework. Disponible en Internet: <<http://static.springsource.org/spring/docs/2.0.x/reference/mvc.html>>.

[TERRACOTTA WORKS] Terracotta Inc. *Terracotta Documentation* [en línea]. [San Francisco, California, Estados Unidos]. 2010 [ref. de julio 2011 a julio de 2012]. Disponible en: <<http://terracotta.org/documentation/overview>>

[TERRACOTTA VENTAJAS TOOLKIT] Terracotta Inc. *Migrating from Terracotta DSO* [en línea]. [San Francisco, California, Estados Unidos]. 2010 [ref. de julio 2011 a julio de 2012]. Disponible en: <<http://docs.terracotta.org/confluence/display/docs/Migrating+From+Terracotta+DSO>>

[TERRACOTTA SESSIONS] Terracotta Inc. *WebSessions Reference* [en línea]. [San Francisco, California, Estados Unidos]. 2010 [ref. de julio 2011 a julio de 2012]. Disponible en: <<http://terracotta.org/documentation/web-sessions/reference-guide>>

[TERRACOTTA EHCACHE] Terracotta Inc. *Distributed EhCache Configuration Guide* [en línea]. [San Francisco, California, Estados Unidos]. 2010 [ref. de julio 2011 a julio de 2012]. Disponible en: <<http://terracotta.org/documentation/enterprise-ehcache/configuration-guide>>

[TERRACOTTA TOOLKIT] Terracotta Inc. *Working with the Terracotta Toolkit* [en línea]. [San Francisco, California, Estados Unidos]. 2010 [ref. de julio 2011 a julio de 2012]. Disponible en: <<http://terracotta.org/documentation/terracotta-toolkit/toolkit-usage>>