

Índice de contenido

1. Abstract.....	2
2. Motivación / Propósito.....	2
3. Estado del arte.....	3
4. Plan de la obra.....	4
4.1. Calendario.....	4
4.2. Metodología.....	4
5. Desarrollo.....	6
5.1. Introducción.....	6
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.....	13
5.3.3. Arquitectura física propuesta.....	17
5.4. Descripción de las herramientas.....	19
5.4.1. JBoss AS.....	19
5.4.1.1. Herramientas similares.....	19
5.4.2. Terracotta.....	20
5.4.2.1. Herramientas similares.....	21
5.5. Pruebas.....	21
5.6. Problemas encontrados y soluciones propuestas.....	34
5.6.1. Generales.....	34
5.6.2. JBoss.....	45
5.6.3. Terracotta.....	49
5.7. Resultados.....	51
5.7.1. JBoss.....	52
5.7.2. Terracotta.....	60
6. Conclusiones.....	68
7. Datos de los autores.....	69
Barrabino, Diego.....	69
Moreyra, Martín.....	73
8. Bibliografía.....	77

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 casi bajo cualquier condición, 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 online 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 construidos 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 el mismo 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.

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): propone 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 esta misma librería. 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]

4. Plan de la obra

4.1. Calendario

Completar

4.2. Metodología

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

- 1.- Concepción (4 semanas de duración)
- 2.- Elaboración (4 semanas de duración)
- 3.- Análisis de la arquitectura (12 semanas de duración)
- 4.- Construcción (8 semanas de duración)
- 5.- Finalización (4 semanas de duración)

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

A lo largo de todo el proyecto, se administrará un informe general e integrador, y particularmente sobre el fin de cada sección se desarrollarán los informes correspondientes que den cuenta que se llevó a cabo dicha etapa, qué se hizo, qué se esperaba conseguir, qué se consiguió, etc., según la naturaleza de cada sección. Cada una de estos informes se adosará al informe general e integrador, de manera tal que termine siendo una sola unidad homogénea.

Se tiene más detalle de esto en la sección Calendario de este informe.

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 de performance vistas en [FIELDING]: buena performance, escalabilidad y alta disponibilidad. Al decidir las características de la aplicación, se eligió 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. Esta decisión se tomó para poder poner a prueba los servicios de sincronización de cada tipo de cluster.

Habiendo implementado la aplicación de pruebas, se la ajustó 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, el framework OpenEJB) 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 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.

A continuación, se definieron las pruebas automáticas a ejecutar sobre cada ambiente. Se definieron pruebas de stress, carga, sólo sincronización, y de operación de todas las funcionalidades de la aplicación de pruebas salvo por las de sincronización.

Posteriormente, estructuramos cada uno de los ambientes con cada uno de los tipos de clusters.

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. Durante un tiempo consideramos montar los nodos en imágenes virtualizadas del sistema operativo, pero debimos descartar esa posibilidad, ya que los recursos de las computadoras utilizadas no eran muy altos (memoria RAM y procesador) y las mismas no soportaron esa combinación.

Finalmente, ejecutamos la aplicación de pruebas en el ambiente de JBoss junto con el software para recolectar estadísticas la cantidad de veces necesaria, hasta que se completaron las pruebas para JBoss, 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.

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.

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.

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. Así como puede darse que los trabajos prácticos pueden ser grupales y colaborativos en su confección, también puede darse que la corrección sea grupal y colaborativa.

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 incluso colaborativamente, 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 toda una universidad con gran cantidad de alumnos, por lo que la carga introducida por cualquiera de los casos 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

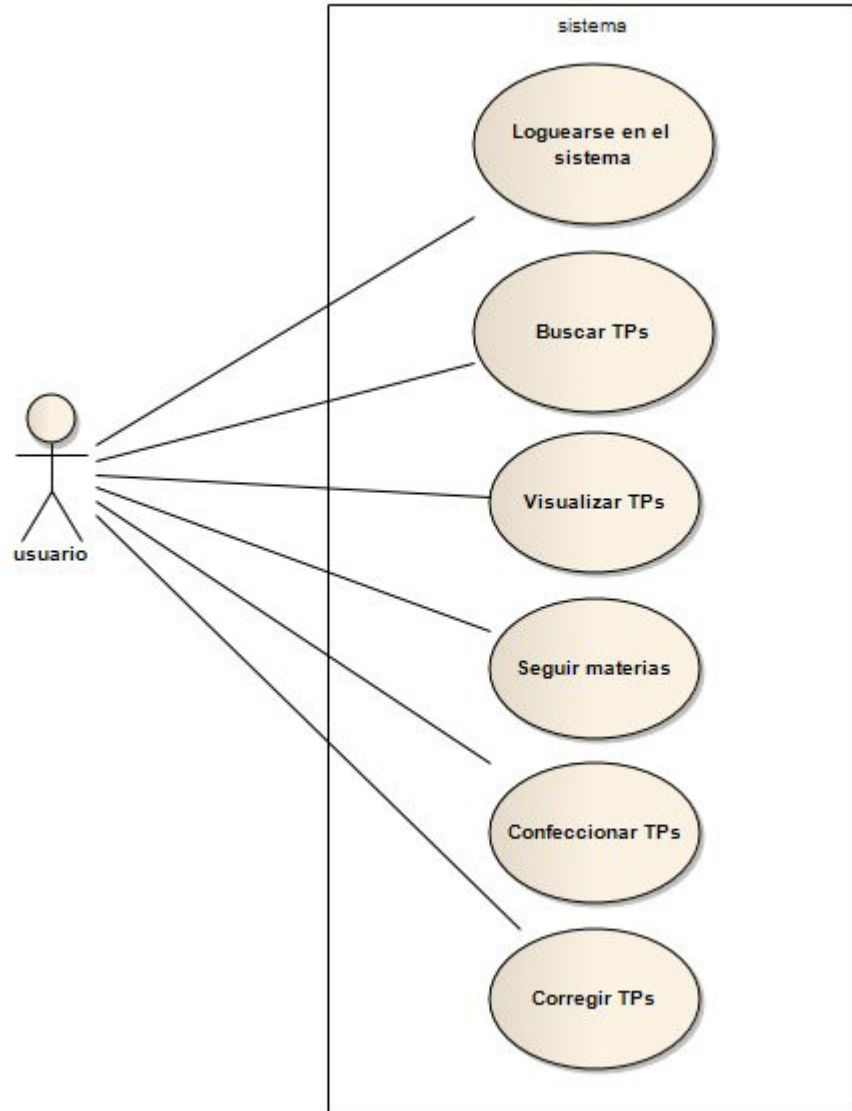
2.- Buscar TPs: pantalla de búsqueda de trabajos según cuatrimestre y materia.

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.

uc 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.

En nuestro caso nos enfocamos en la segunda opción: la escalabilidad horizontal. El objetivo del presente trabajo es justamente la implementación de un cluster, que es básicamente 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

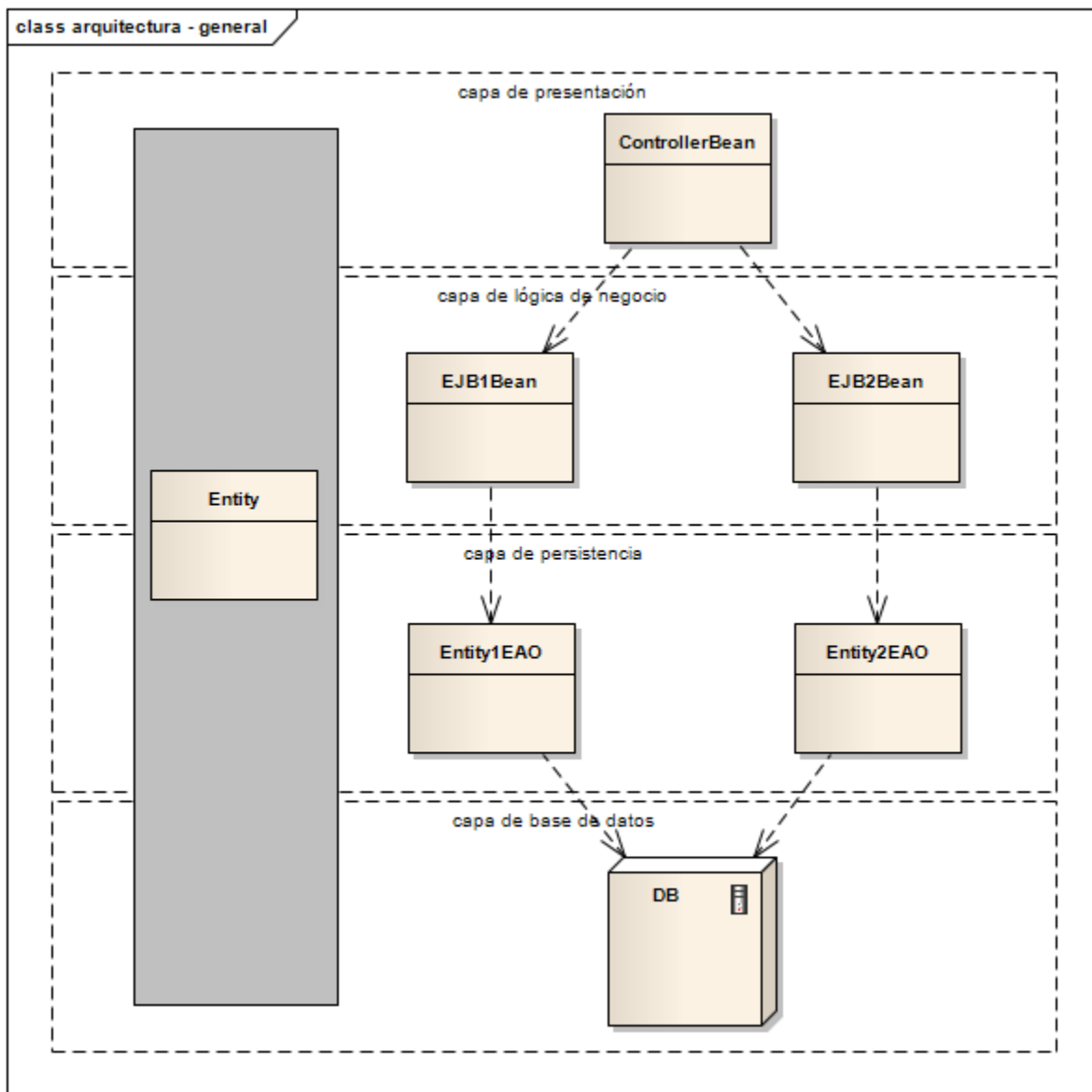
A nivel de arquitectura elegimos una combinación de estilos arquitectónicos que nos proveerán de las propiedades enunciadas en la sección anterior, los estilos son: cliente-servidor en capas + repositorio replicado [FIELDING]. En primer lugar una arquitectura en capas cliente-servidor, significa que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



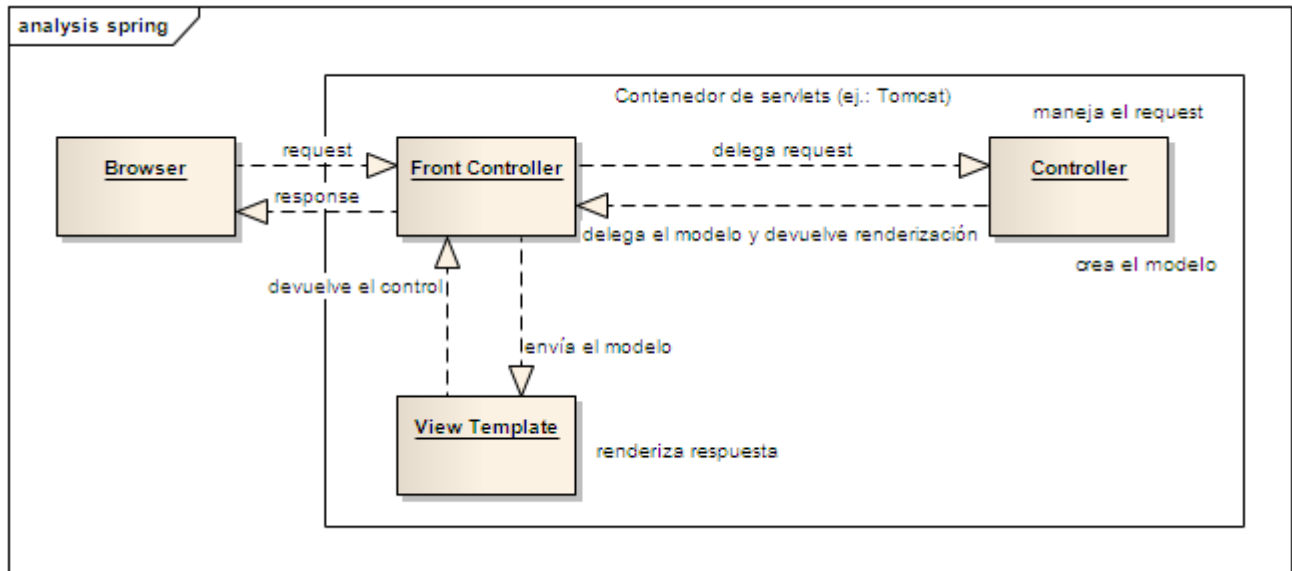
En el gráfico anterior podemos observar la separación en capas, pero también se hace referencia a patrones de diseño específicos utilizados en cada capa.

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

Presentación

La capa de presentación utiliza el patrón de diseño MVC, en particular utiliza el Framework SpringMVC. 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:

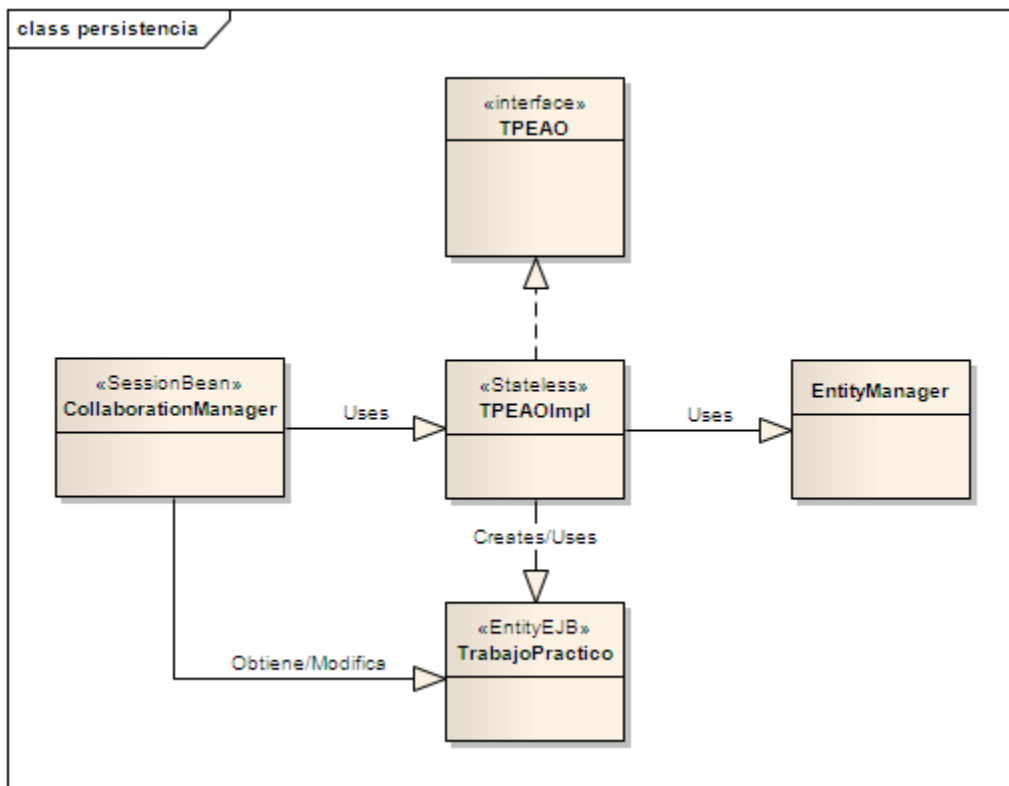


Persistencia

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 está implementado directamente por el framework de persistencia utilizado: Hibernate

- 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.



Como se puede ver de 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.

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.

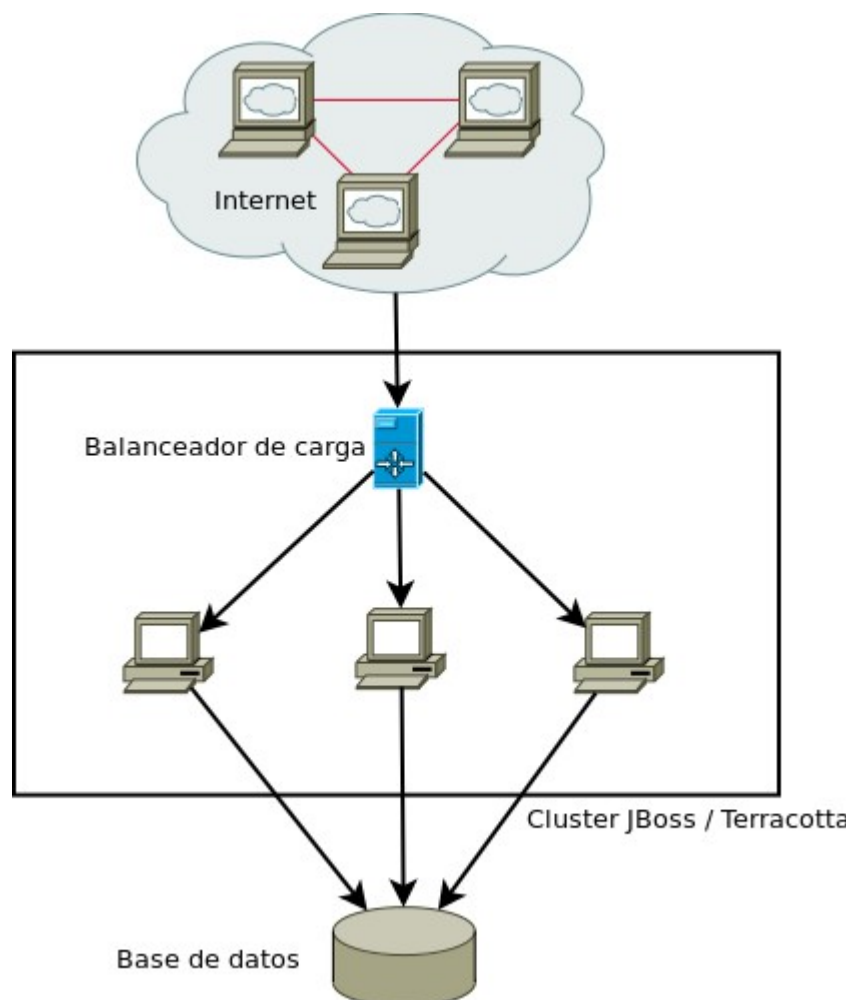
En un contexto ideal se pueden poner en marcha dos nodos, con el fin de tener redundancia en caso de que uno de ellos falle, e ir activando nodos extra a medida que la carga de los mismos vaya aumentando. Por este motivo nos interesa la propiedad de elasticidad mencionada en el punto 5.3.1 de este documento

- 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.



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 en Java puro. Al estar basado en Java, 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.

JBoss 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].

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

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 [TERRACOTTA WORKS].

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].

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.

5.4.2.1. Herramientas similares

Terracotta

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

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 varios 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 variando 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.

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 estándares 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 Lubuntu 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) CPU 2.1 GHZ de cuatro núcleos, placa madre Intel, placa de red, placa de red wifi, memoria RAM de 4 Gb, disco rígido Samsung 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) CPU 2.1 GHZ de cuatro núcleos, placa madre Intel, placa de red, placa de red wifi, memoria RAM de 4 Gb, disco rígido Samsung 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.

Esta computadora Desktop no será utilizada en la demostración del presente trabajo, debido a problemas de transporte; en su lugar se usará, tal como se explicó, la Netbook

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 (PC Desktop)
- Clientes: Utilizamos 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.

Pruebas

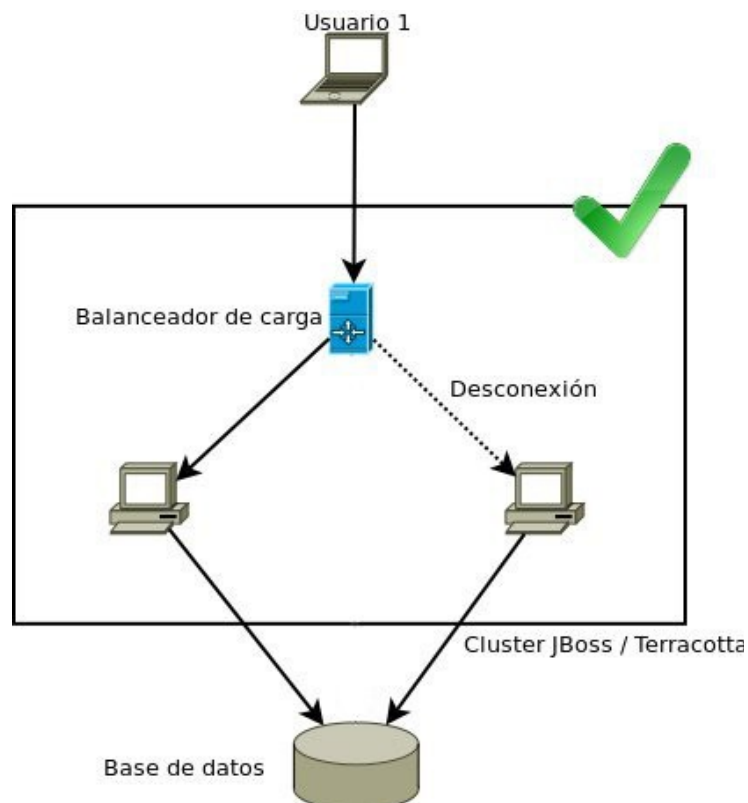
Para armar los casos de prueba, confeccionamos la siguiente planilla, que instanciaremos en cada uno de las pruebas definidas.

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					

Pruebas manuales

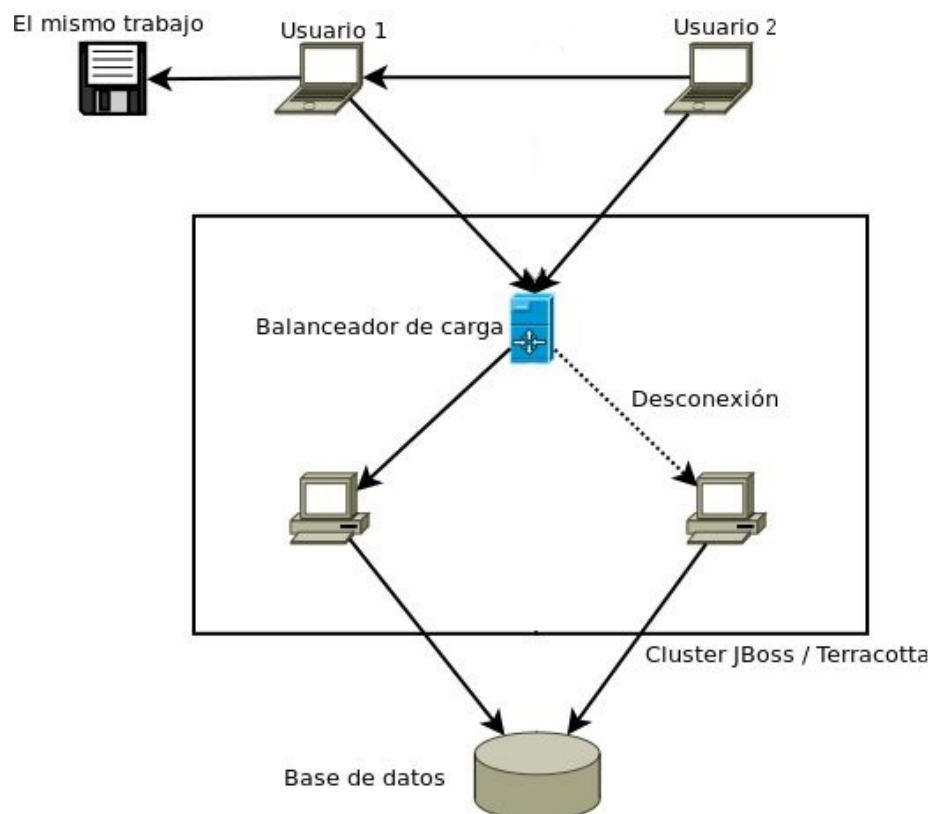
Disponibilidad del sistema

Descripción		<u>Disponibilidad incesante y automática</u>	
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	



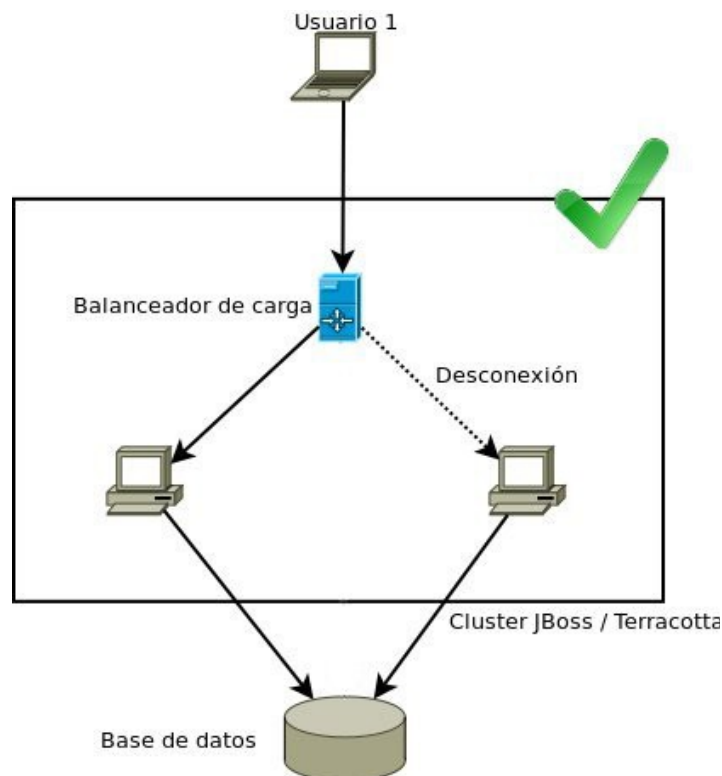
Desconexión de un nodo en esquema colaborativo

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	



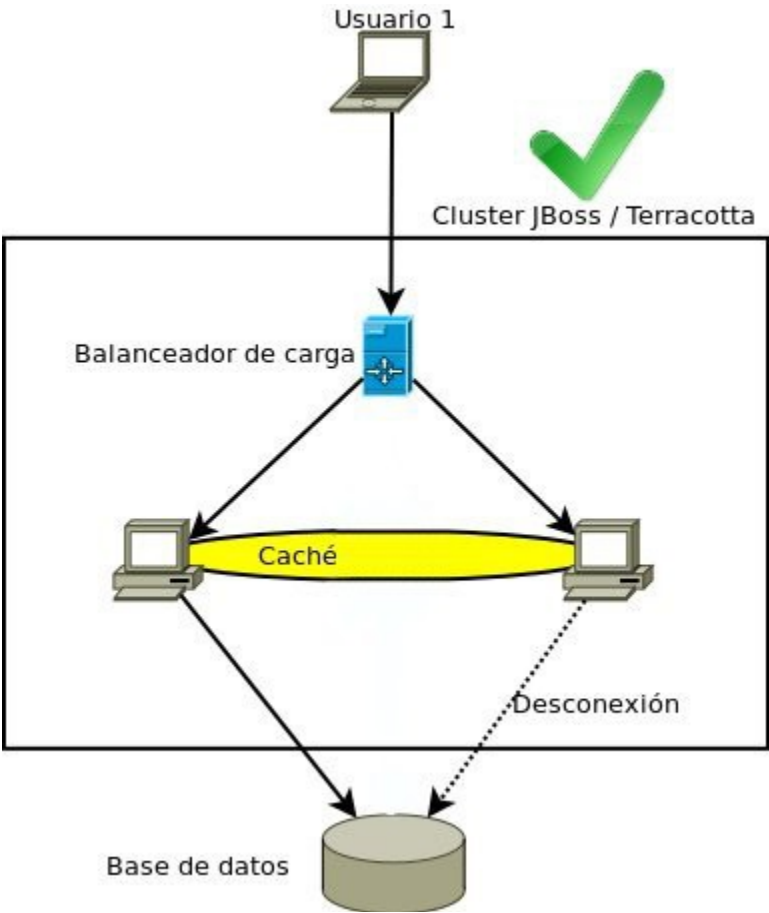
Replicación de Sesión

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.	



Caché

Descripción		Consultar datos <u>cacheables</u> y <u>verificar</u> la <u>replicación</u> de la cache en <u>los</u> <u>diferentes</u> <u>nodos</u>
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	



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

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

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

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

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

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

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

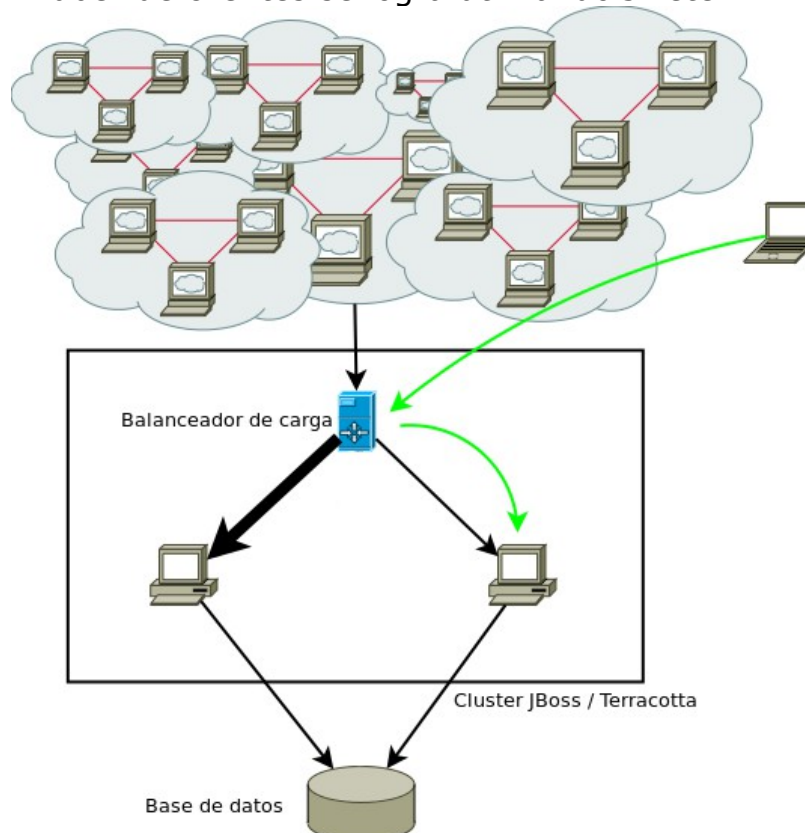
- Sincronizar texto de resolución 10 veces

Load Test Datos No Clusterizados

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	

- 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; etc.

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, se editó 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 logeara información con un nivel de detalle alto, ya que

tampoco queríamos que el costo de logear 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 se editó 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; unblock
with 'mysqladmin flush-hosts'"
```

Esto se dió, 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. Se editó 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, se identificó 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:

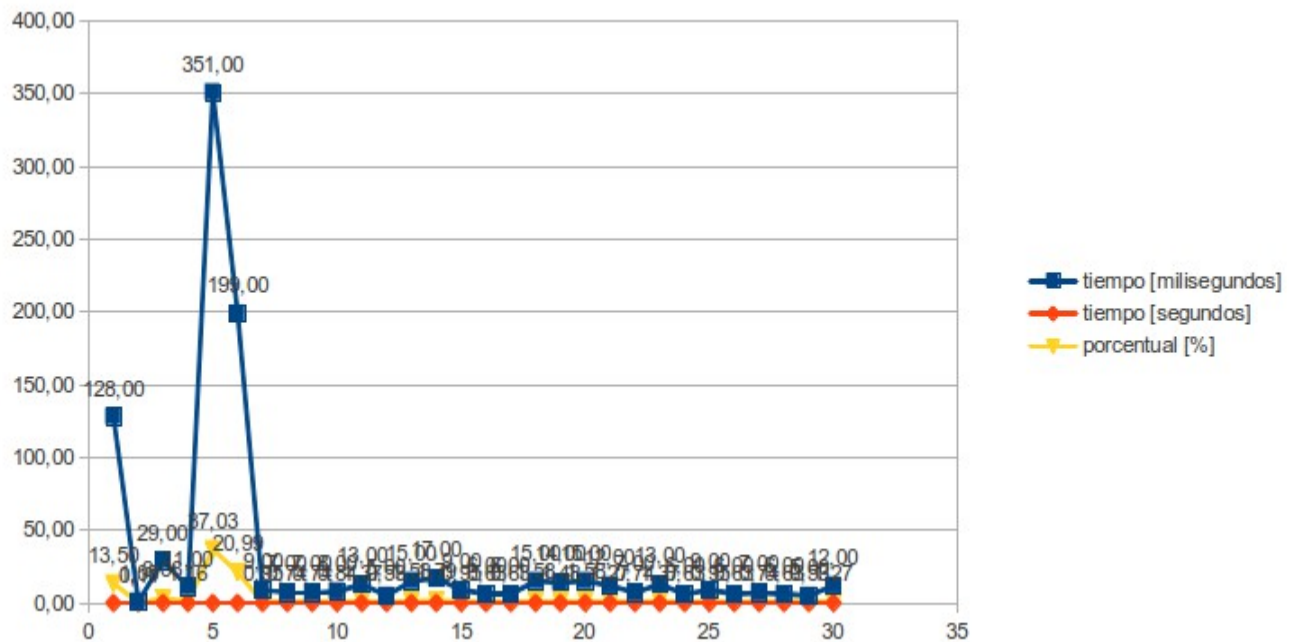
```
session required                pam_limits.so
```

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

Aplicación de prueba

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

En consecuencia, creamos un Profiler, que corrimos varias veces junto con la aplicación, para intentar detectar los problemas. Se adjunta una imagen de una de las corridas del 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
- 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>

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 bckground de recolección cuando esté ocupado más del 60% de la memoria
- XX:+UseCMSInitiatingOccupancyOnly : utilizar sólomente el parámetro anterior para definir cuando levantar el thread del gc

Estos parámetros se utilizan en el entorno productivo del CONICET, y desde CONICET se los ha recomendado a otras instituciones, siempre con éxito, por lo cual consideramos que es buena idea tomarlos para nuestro trabajo

Servidores

Todos los servidores se levantan con el modo debug en off

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, 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

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 determinar si el cluster seguía en funcionamiento o no

localhost/balancer-manager?b=ajpcluster&w=ajp://worker2:8109&nonce=1b32a629-f89f-4b63-b8fe-b207c0957530

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	4	0	1.8K
------------------------------------	---------	--	---	---	----	---	---	------

Edit worker settings for ajp://worker2:8109

Load factor:

LB Set:

Route:

Route Redirect:

Status: Disabled: ☐ | Enabled: ☒

Apache/2.2.22 (Ubuntu) Server at localhost Port 80

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.

5.6.2. 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

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 a 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 datos clusterizados

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 tratamos de implementar, al igual que en el punto anterior, una caché de entidades distribuida.

Continuando con el escenario incómodo explicado en el punto anterior, cabe destacar que nuevamente la documentación del JBoss 5 es superior en este aspecto a la documentación de JBoss 6, debido a que el JBossCache de JBoss 5

tiene un módulo llamado PojoCache, ideal para este uso. En JBoss 6, Infinispan, el reemplazo de JBossCache, no presenta un módulo equivalente

En consecuencia, se debió configurar a mano el PojoCache en JBoss 6, y por suerte al igual que en el caso anterior tuvimos éxito: en el cluster con JBoss, las sincronizaciones se llevan a cabo por PojoCache

La configuración se almacena en 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>
```

Además, se creó en el proyecto una clase de nombre PojoCacheManager, que lo que hace es:

- Cuando se levanta un nodo, anula el registro de los datos únicos a sincronizar que aporta ese nodo
- Identifica al nodo que se levantó primero que todos
- En ese nodo, vuelve a registrar los datos únicos a sincronizar
- A partir de ese momento, cuando cualquier nodo quiera consultar los datos sincronizables, se comunican con el nodo que sí los tiene registrados
- Si el nodo que tiene registrados los datos sincronizables se cae, automáticamente se registran esos datos sin perder valores en el nodo que se levantó en segundo lugar; y en caso de más caídas, así siguiendo mientras existan nodos

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

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, 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</pro  
perty>
```

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 datos clusterizados

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.

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:

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

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

Replicación de sesiones

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

```
<Valve  
className="org.terracotta.session.TerracottaTomcat70xSessionValv  
e" 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.

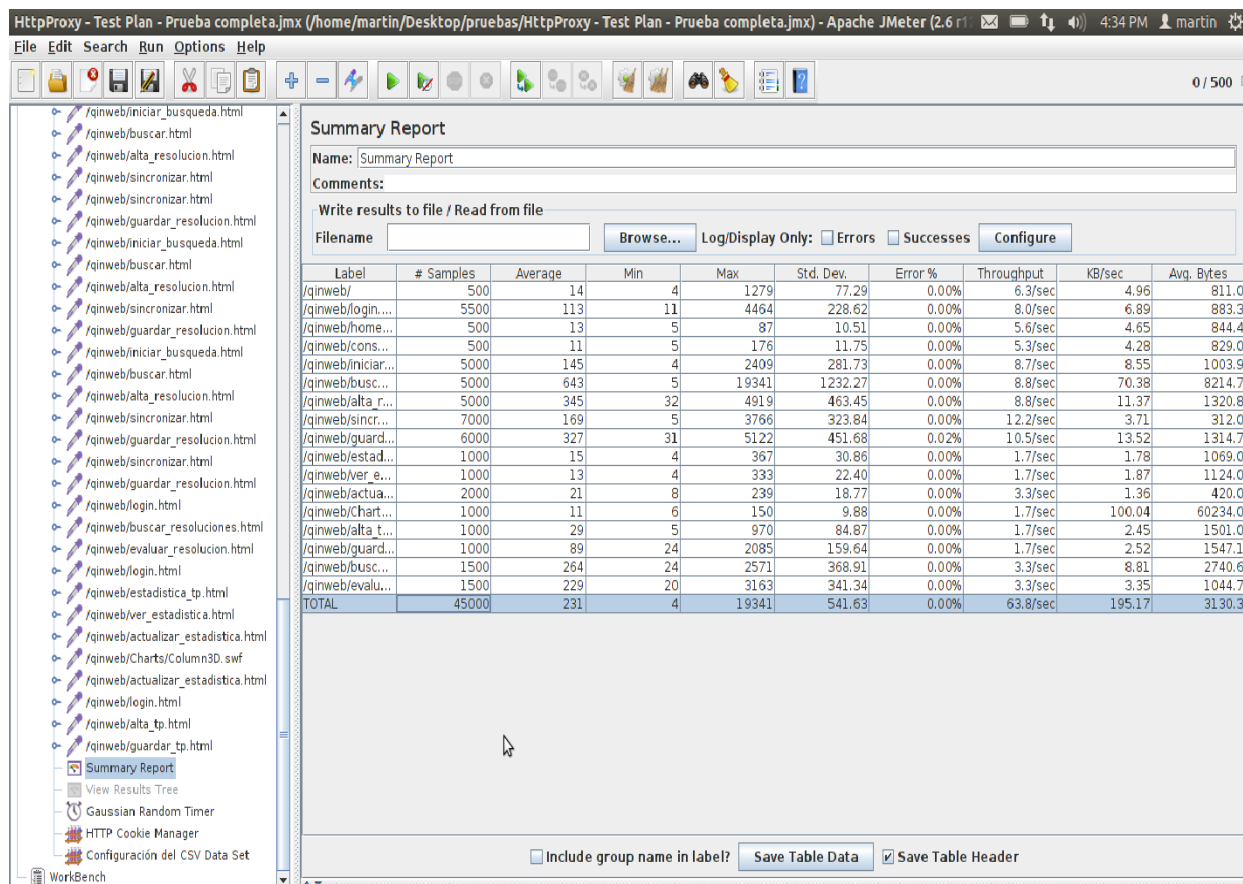
Array de Servidores Terracotta

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:



The screenshot shows the Apache JMeter Summary Report window. The left sidebar lists various test elements, and the main area displays a table of results. The table includes columns for Label, # Samples, Average, Min, Max, Std. Dev., Error %, Throughput, KB/sec, and Avg. Bytes. The data is organized by test element, with a total row at the bottom.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
/qinweb/	500	14	4	1279	77.29	0.00%	6.3/sec	4.96	811.0
/qinweb/login...	5500	113	11	4464	228.62	0.00%	8.0/sec	6.89	883.3
/qinweb/home...	500	13	5	87	10.51	0.00%	5.6/sec	4.65	844.4
/qinweb/cons...	500	11	5	176	11.75	0.00%	5.3/sec	4.28	829.0
/qinweb/iniciar...	5000	145	4	2409	281.73	0.00%	8.7/sec	8.55	1003.9
/qinweb/busc...	5000	643	5	19341	1232.27	0.00%	8.8/sec	70.38	8214.7
/qinweb/alta r...	5000	345	32	4919	463.45	0.00%	8.8/sec	11.37	1320.8
/qinweb/sincr...	7000	169	5	3766	323.84	0.00%	12.2/sec	3.71	312.0
/qinweb/guard...	6000	327	31	5122	451.68	0.02%	10.5/sec	13.52	1314.7
/qinweb/estad...	1000	15	4	367	30.86	0.00%	1.7/sec	1.78	1069.0
/qinweb/ver_e...	1000	13	4	333	22.40	0.00%	1.7/sec	1.87	1124.0
/qinweb/actua...	2000	21	8	239	18.77	0.00%	3.3/sec	1.36	420.0
/qinweb/login.html	1000	11	6	150	9.88	0.00%	1.7/sec	100.04	60234.0
/qinweb/buscar_resolucion.es.html	1000	29	5	970	84.87	0.00%	1.7/sec	2.45	1501.0
/qinweb/evaluar...	1000	89	24	2085	159.64	0.00%	1.7/sec	2.52	1547.1
/qinweb/login.html	1500	264	24	2571	368.91	0.00%	3.3/sec	8.81	2740.6
/qinweb/estadistica_tp.html	1500	229	20	3163	341.34	0.00%	3.3/sec	3.35	1044.7
/qinweb/ver_estadistica.html									
/qinweb/actualizar_estadistica.html									
/qinweb/Charts/Column3D.swf									
/qinweb/actualizar_estadistica.html									
/qinweb/login.html									
/qinweb/alta_tp.html									
/qinweb/guardar_tp.html									
SUMMARY REPORT									
TOTAL	45000	231	4	19341	541.63	0.00%	63.8/sec	195.17	3130.3

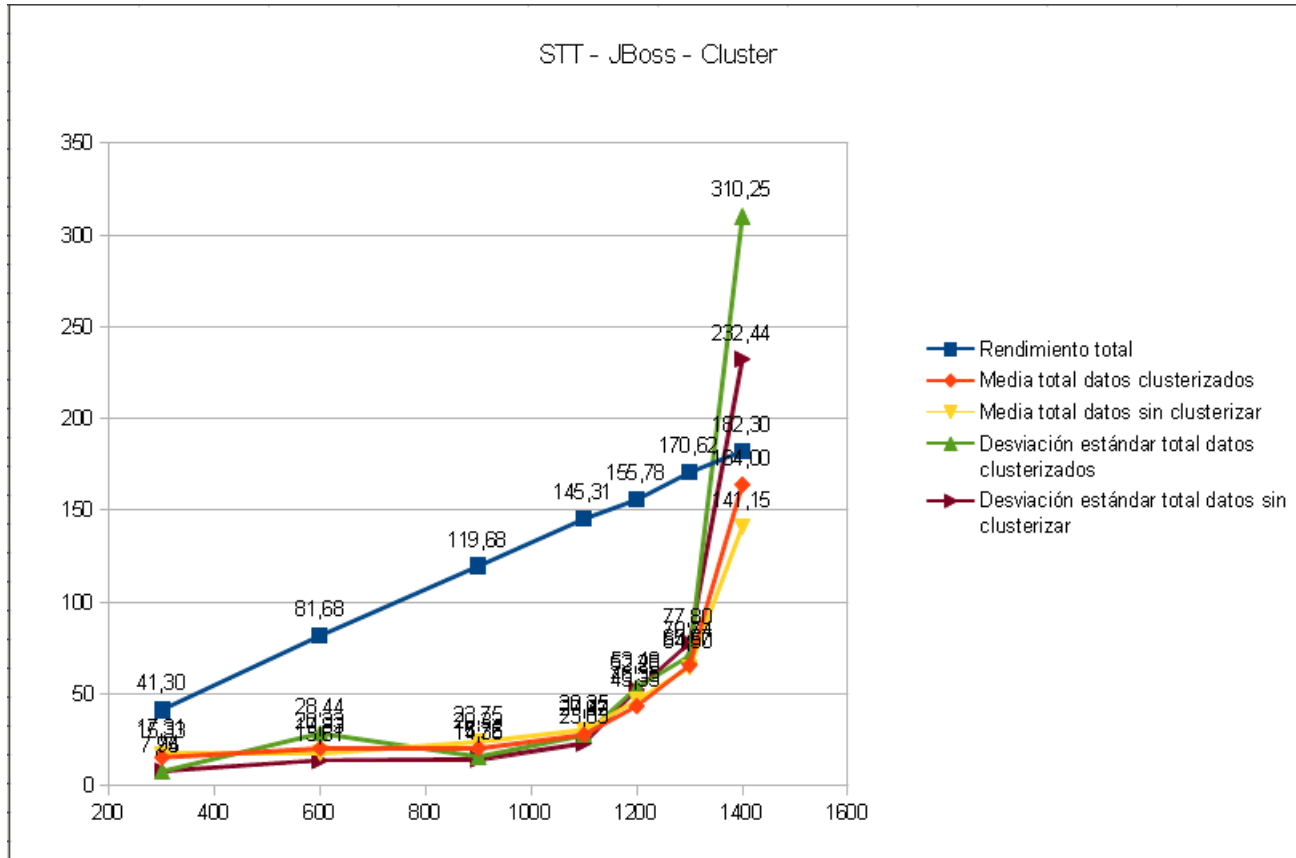
Podemos ver a la derecha las peticiones que se realizan al sistema, y a la izquierda en forma de tabla podemos observar la cantidad de peticiones de cada tipo, los tiempos medios de respuesta, las desviaciones estándares, porcentaje de error, rendimiento y mas.

En todos los casos:

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

5.7.1. JBoss

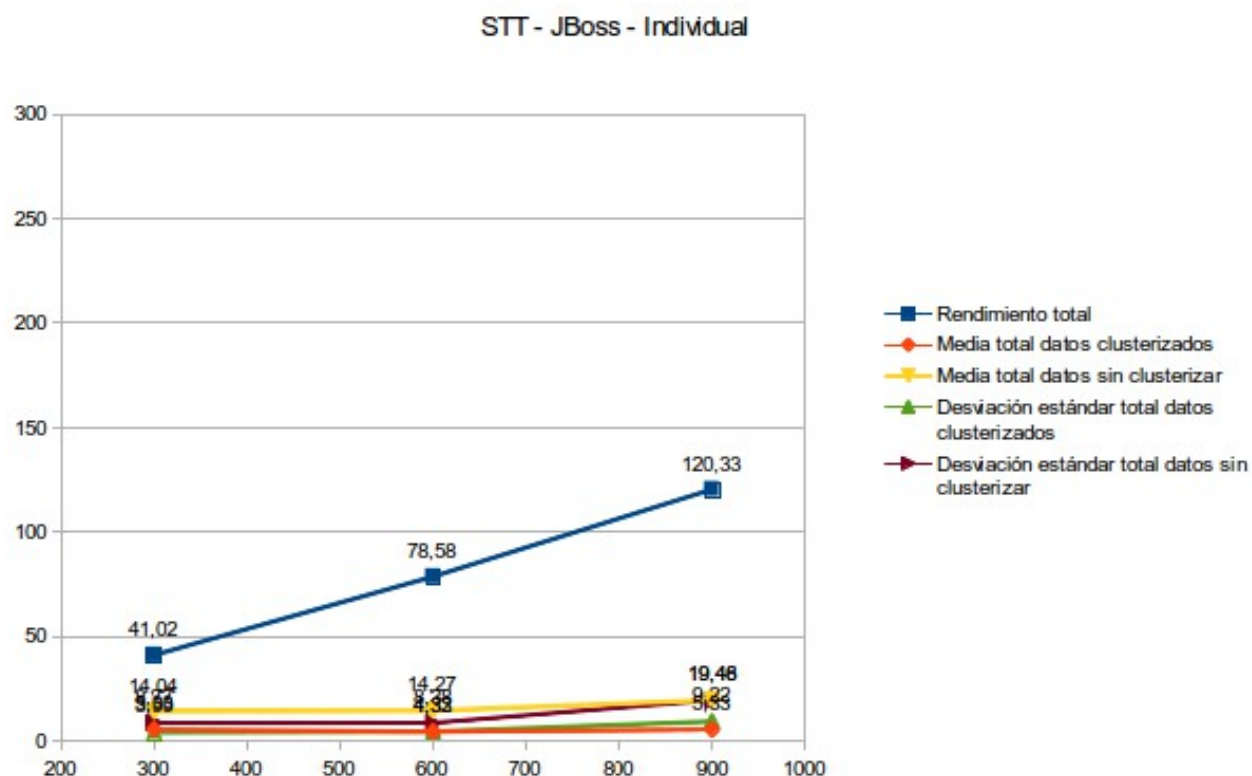
Stress Test - Cluster



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) se está perfectamente dentro de los tiempos estándar de funcionamiento. 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.

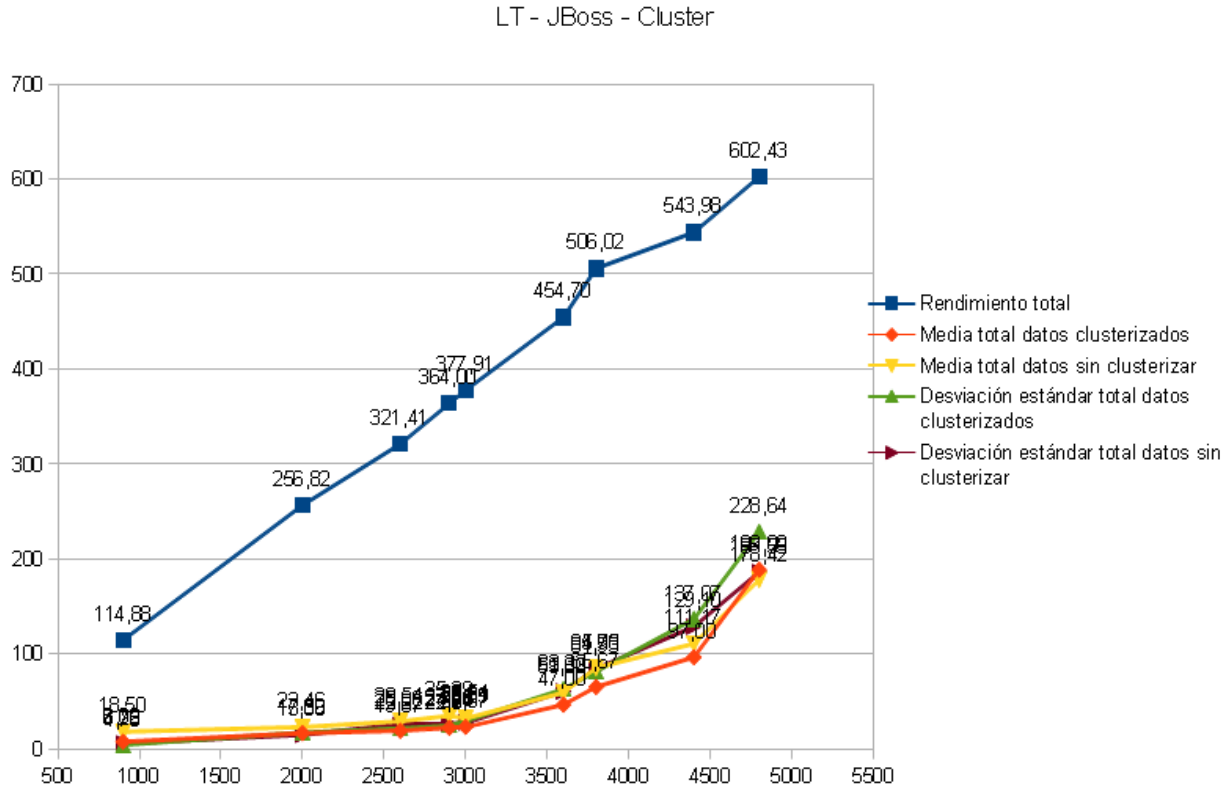
Stress Test - Individual



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

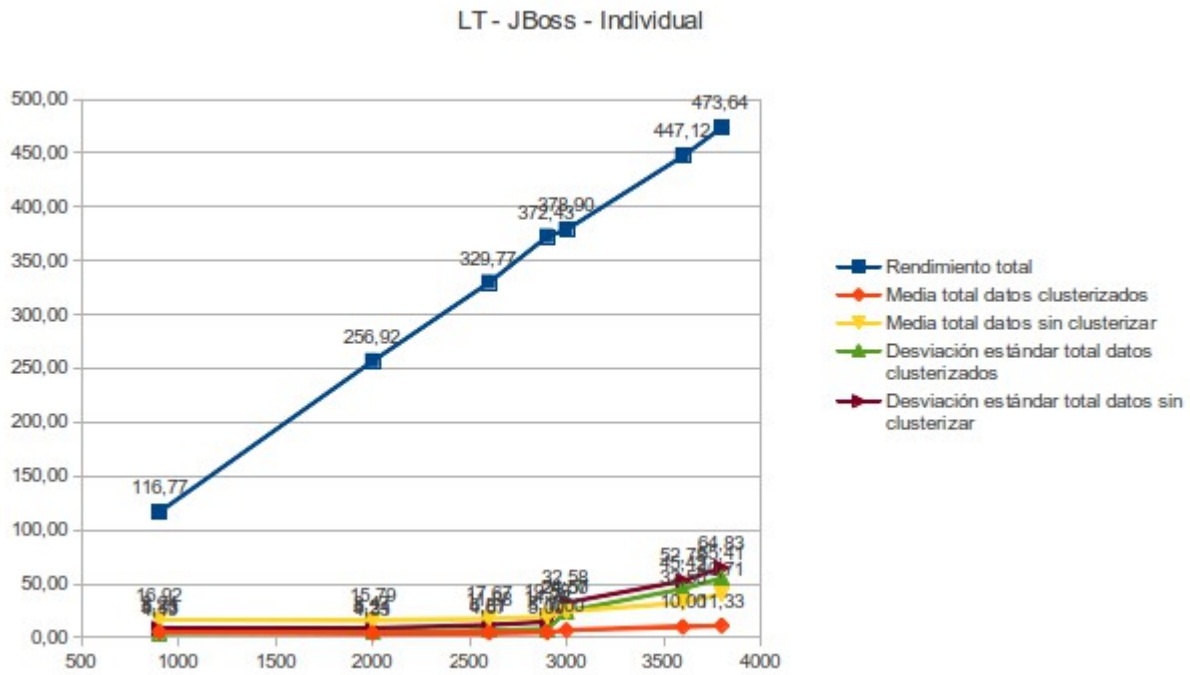
Se puede apreciar aquí una particularidad de la comparación nodo individual - cluster: mientras responde el nodo individual, es más rápido que el cluster, sin embargo, el cluster tiene más escalabilidad y por definición brinda alta disponibilidad.

Load Test - Cluster



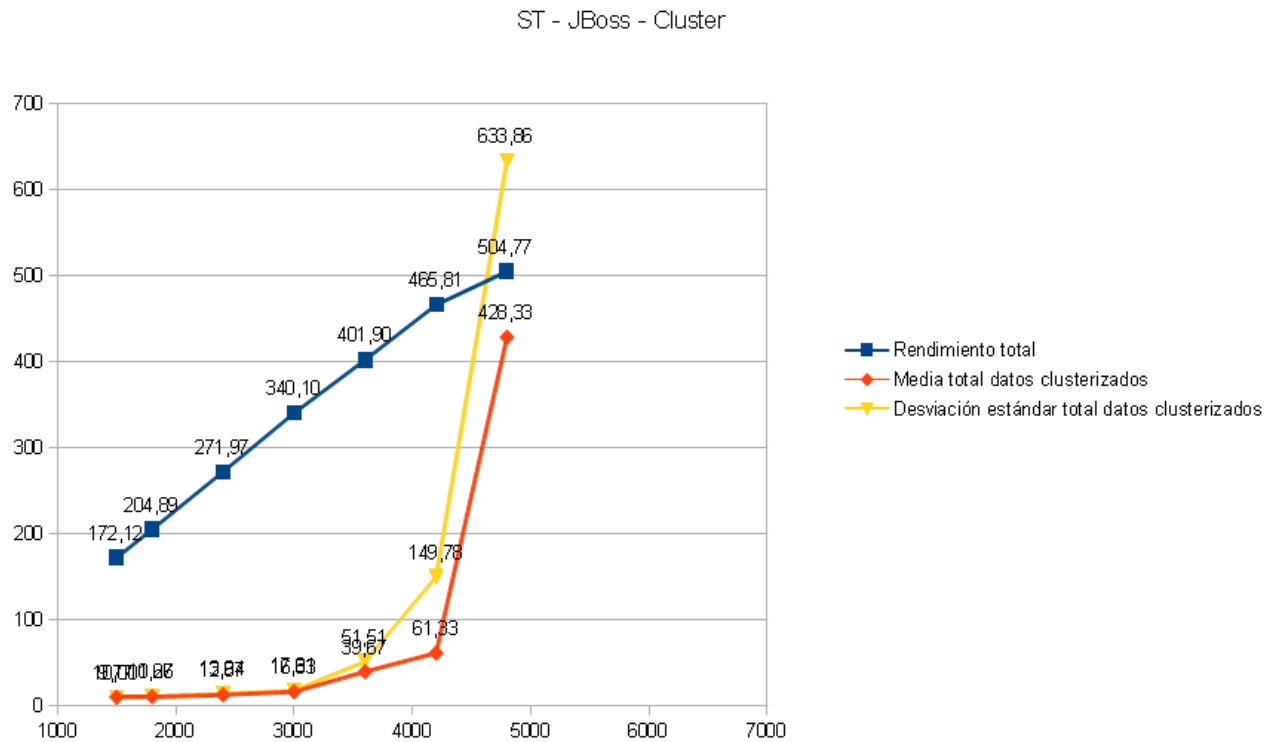
Se puede ver que hasta un poco antes de los 4500 usuarios, se está dentro de los tiempos estándar de respuesta, y que incluso aunque lamentablemente el servicio se separe de esos tiempos de respuesta, de alguna manera llega a atender hasta a 5000 usuarios; a continuación, el ambiente se queda sin recursos, y colapsa

Load Test - Individual



Mismo patrón de comportamiento que en el caso anterior

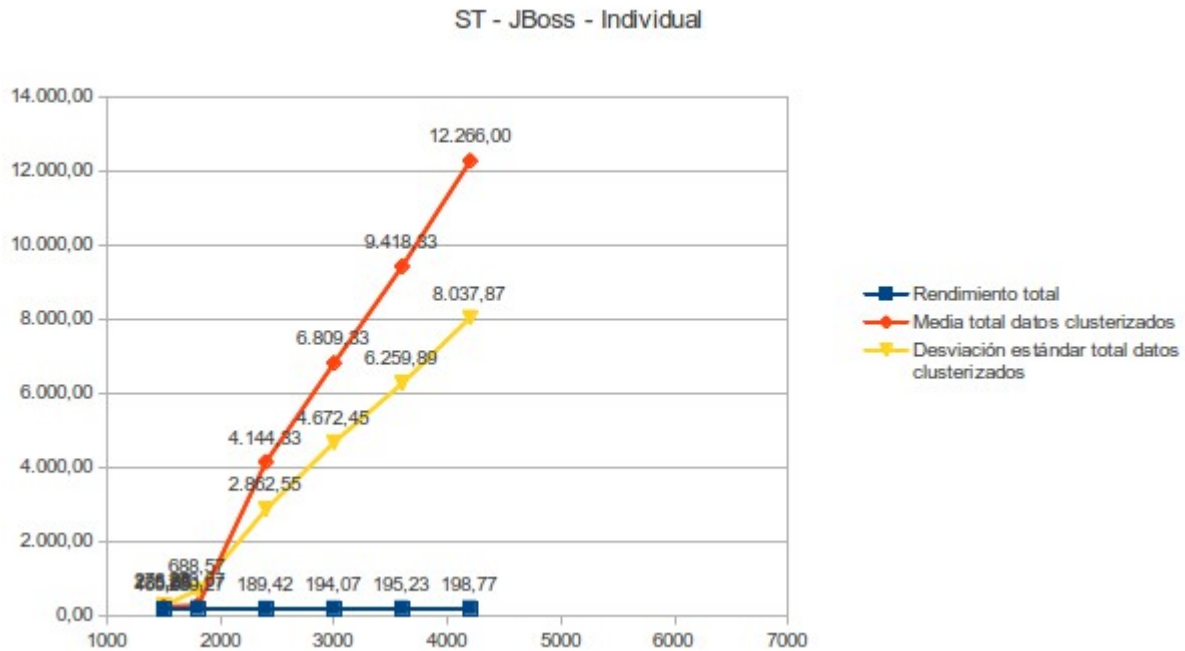
Sync Test - Cluster



Se puede ver que hasta un poco antes de los 4500 usuarios, se está dentro de los tiempos estándar de respuesta, y que incluso aunque lamentablemente el servicio se separe de esos tiempos de respuesta, de alguna manera llega a atender hasta a 5000 usuarios; a continuación, el ambiente se queda sin recursos, y colapsa

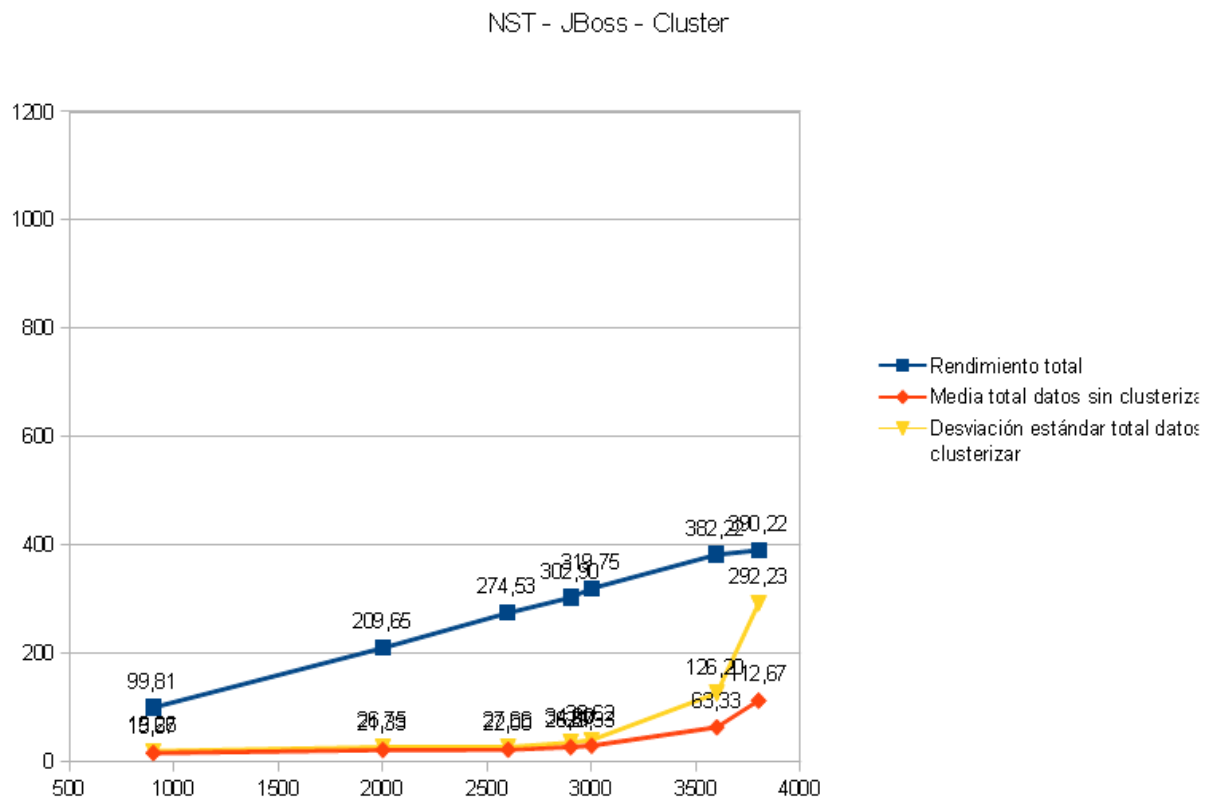
Muy similar al caso de Load Test

Sync Test - Individual



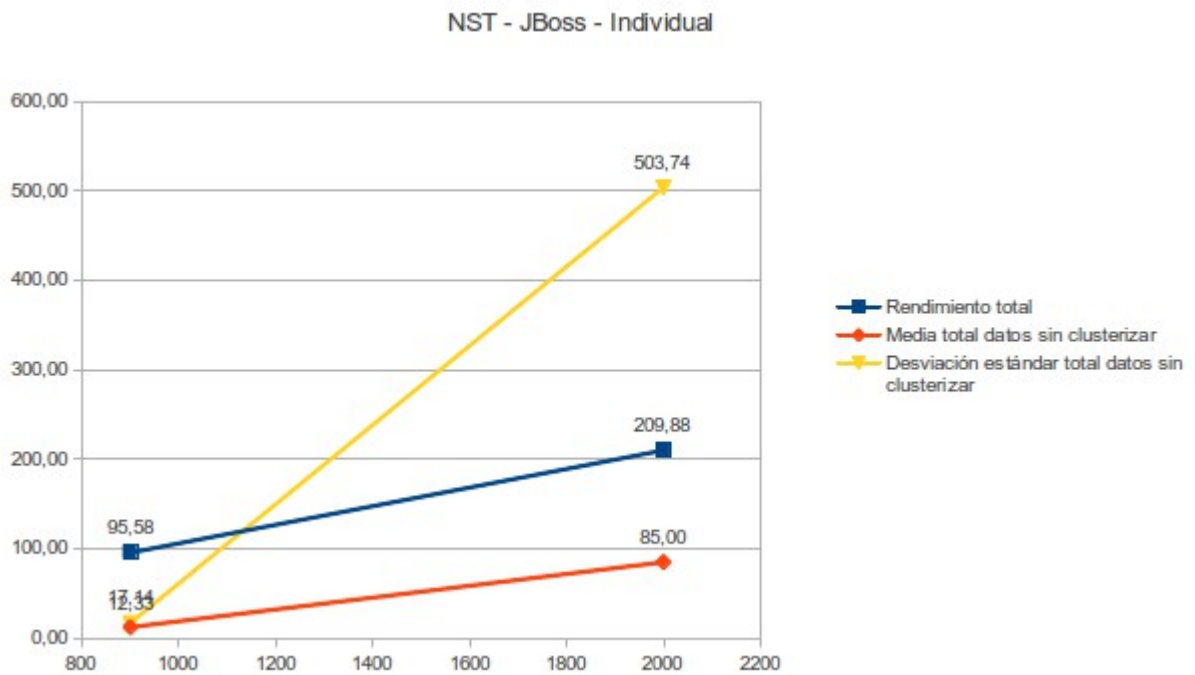
Sencillamente, esta prueba es excesivamente exigente para un solo nodo, independientemente de que en ese único nodo se encuentre un servidor EE como JBoss. Un ejemplo muy claro del funcionamiento superador de un cluster, en este caso.

No Sync Test - Cluster



Hasta aproximadamente los 3500 usuarios, se está dentro de los tiempos de respuesta estándar; 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

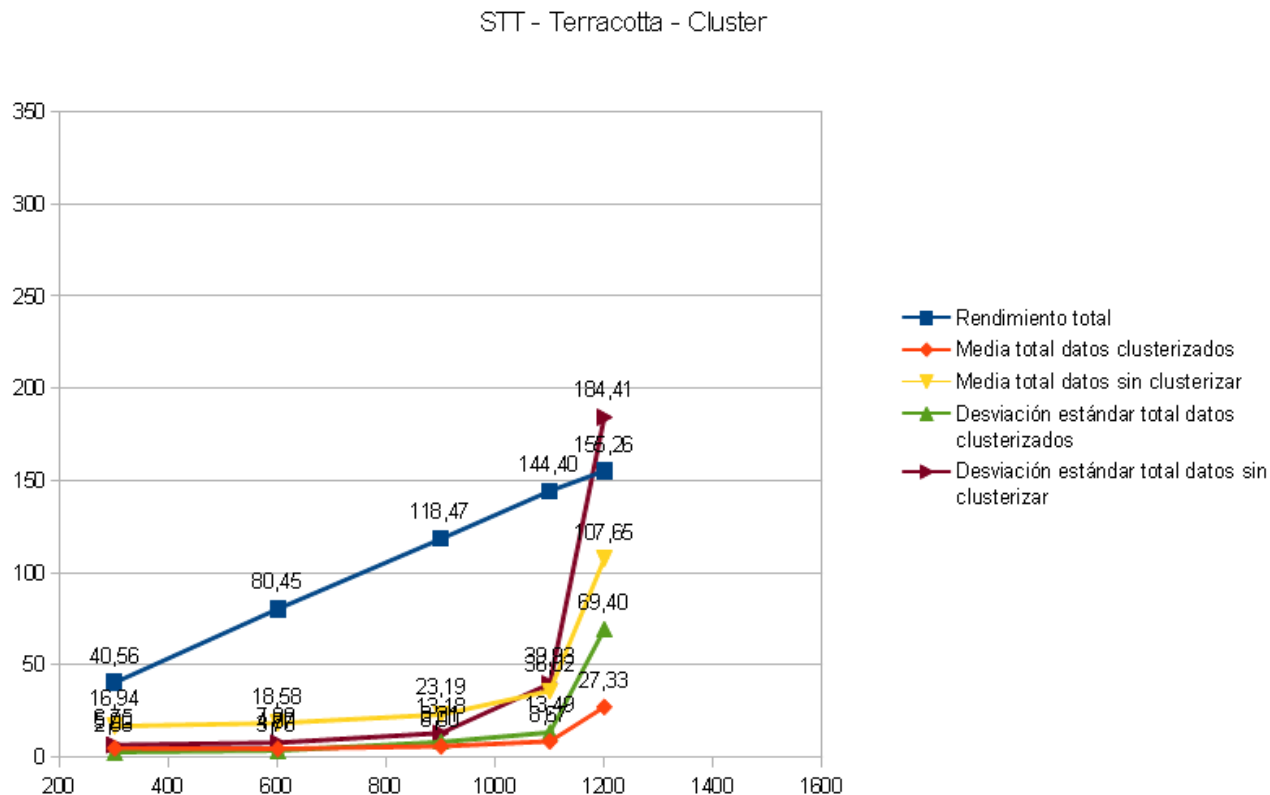
No Sync Test - Individual



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

5.7.2. Terracotta

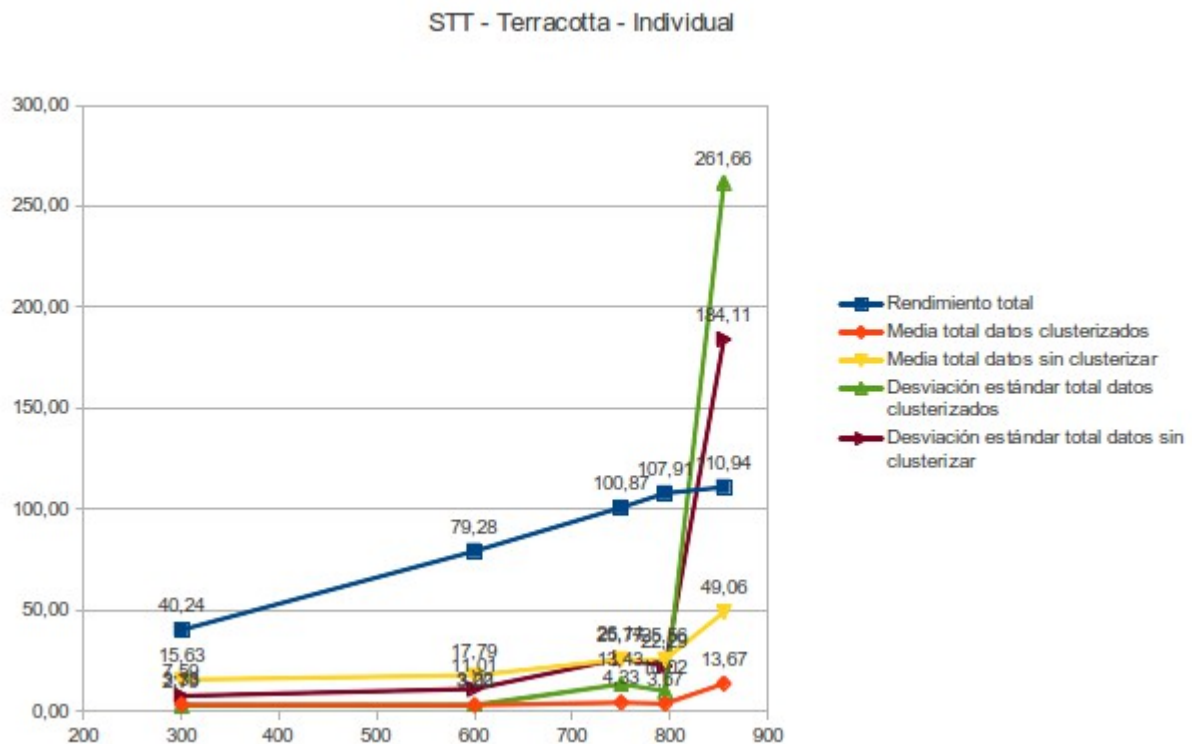
Stress Test - Cluster



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*).

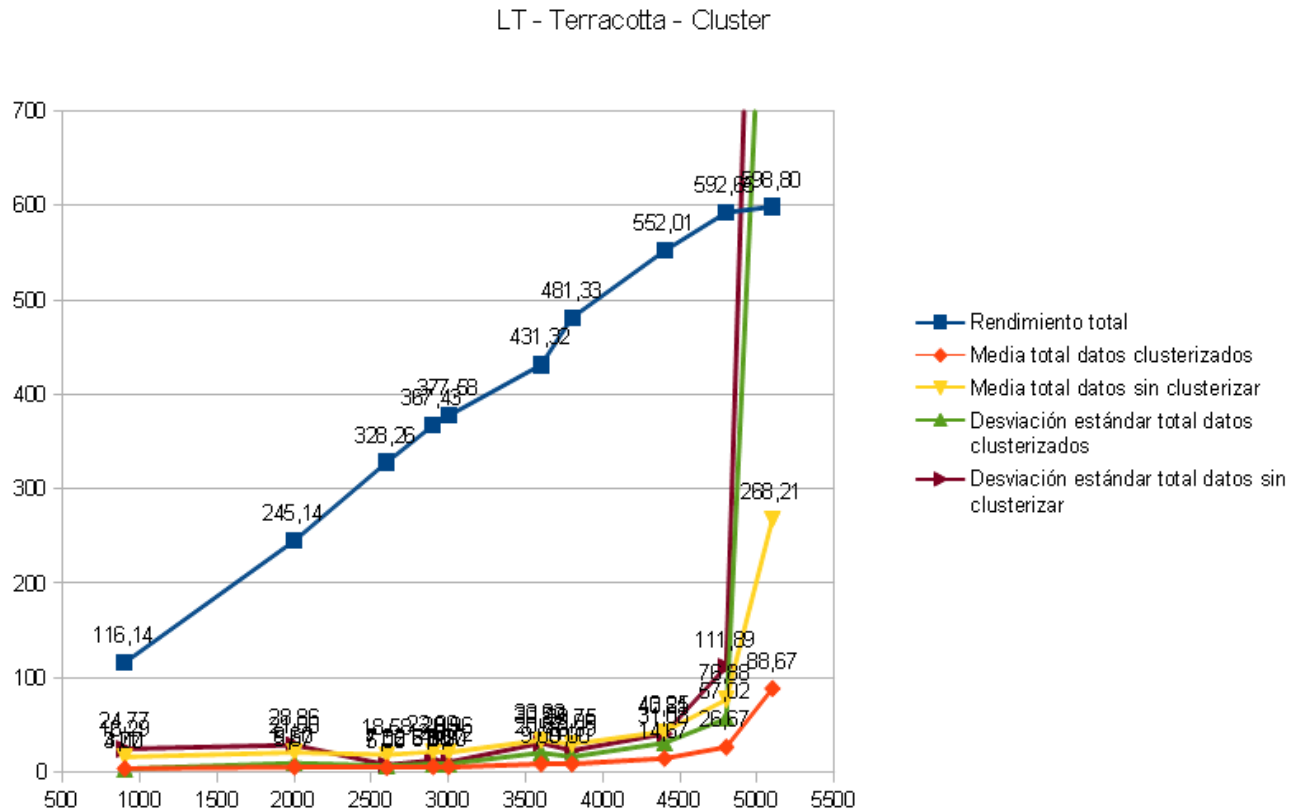
Stress Test - Individual



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

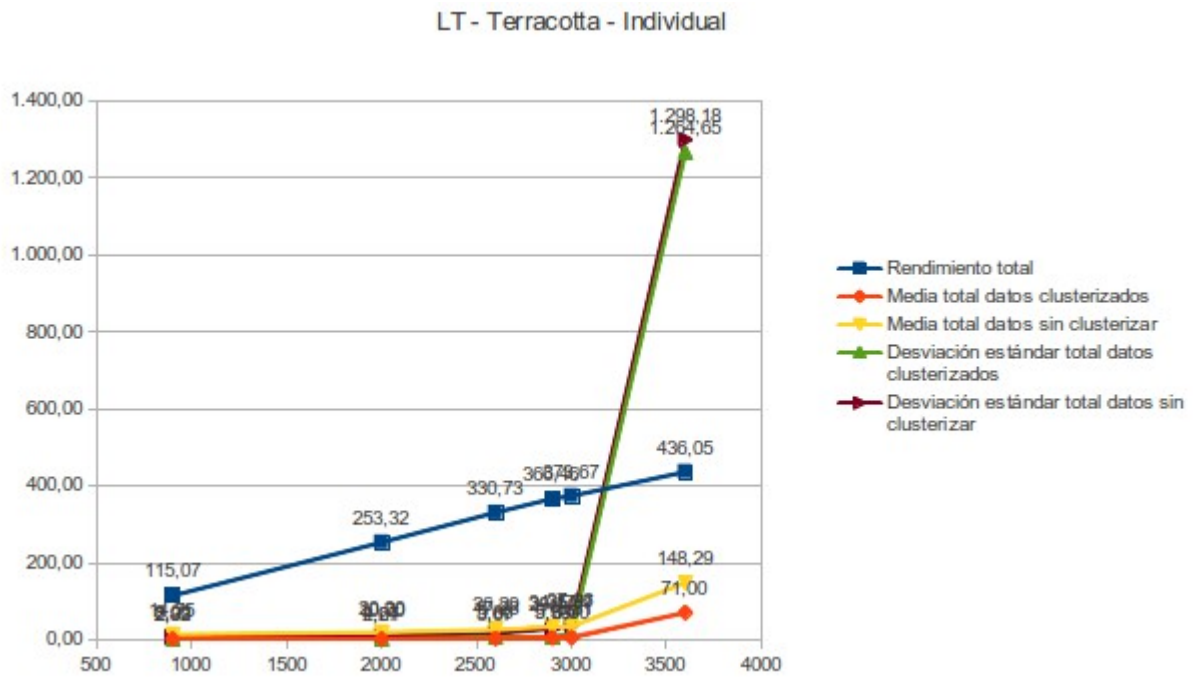
Se percibe también que en este caso, no se llegan a atender a tantos usuarios como en la prueba equivalente en el JBoss individual

Load Test - Cluster



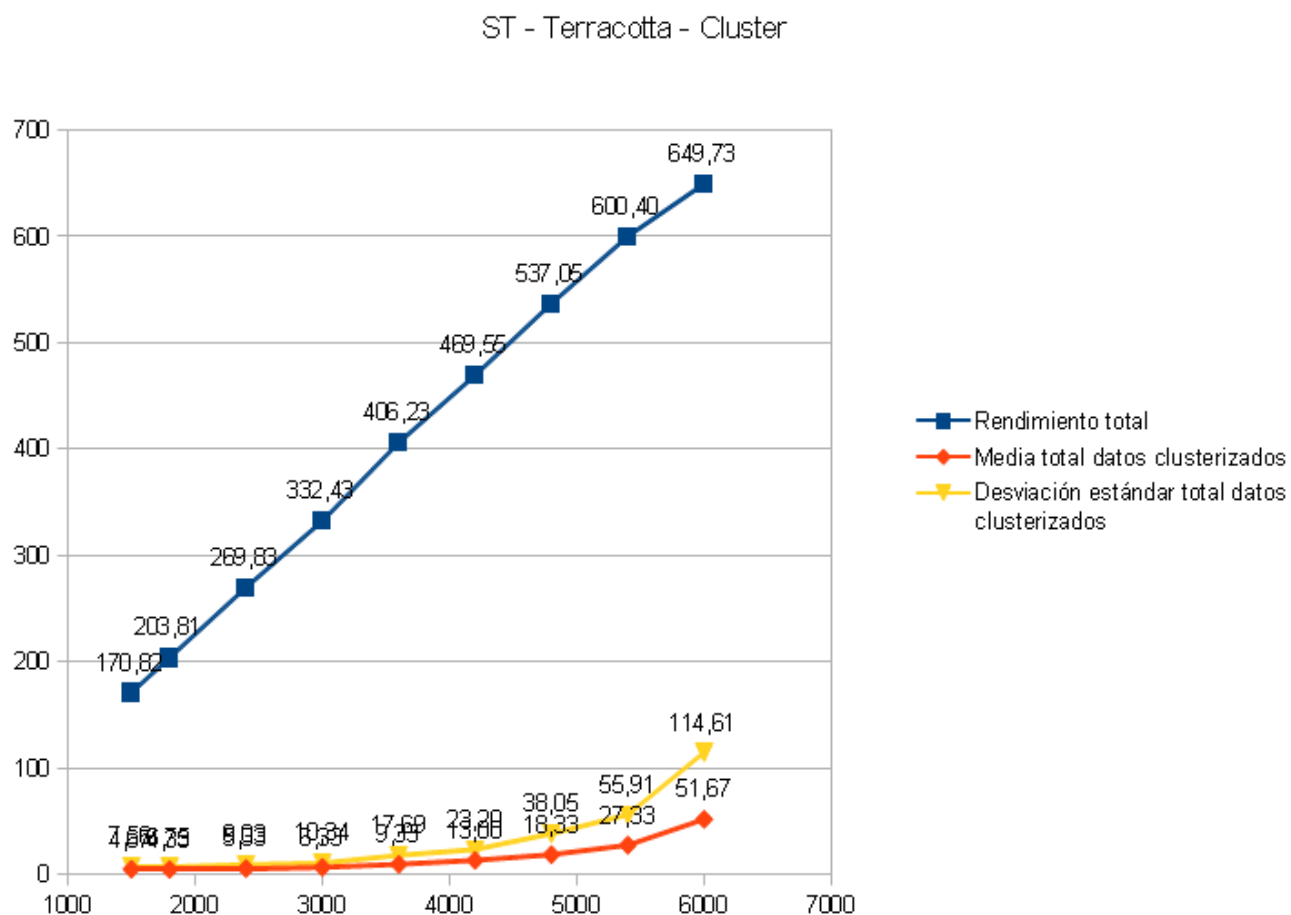
Se repiten todos los patrones observados.

Load Test - Individual



Se repiten todos los patrones observados.

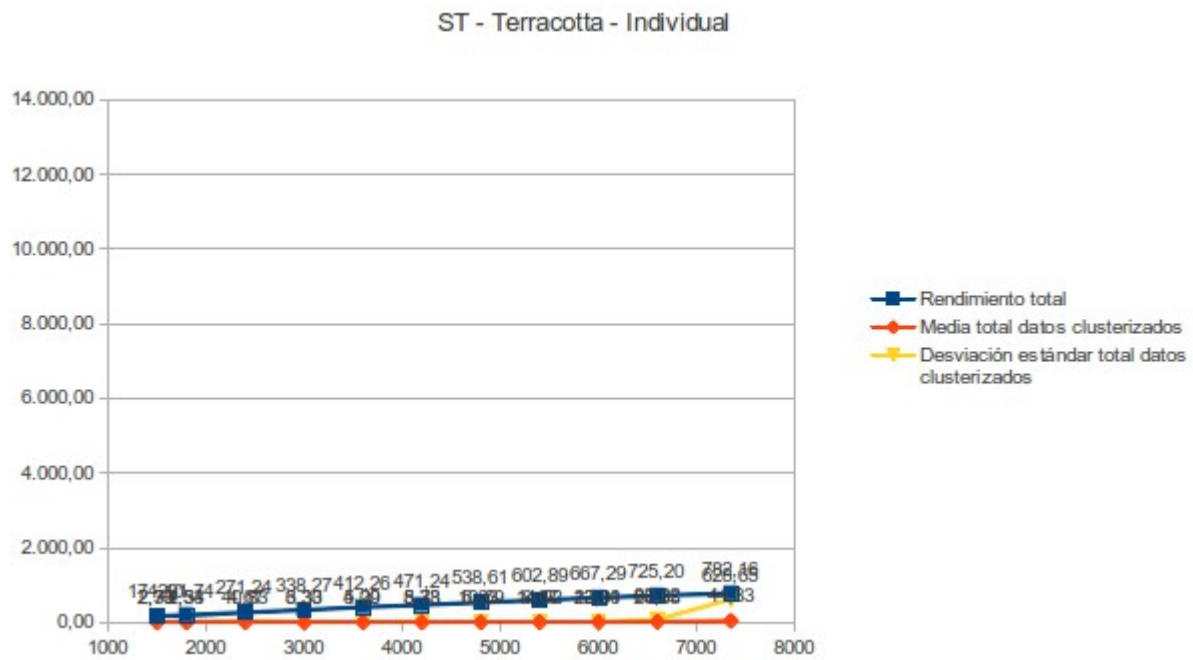
Sync Test - Cluster



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

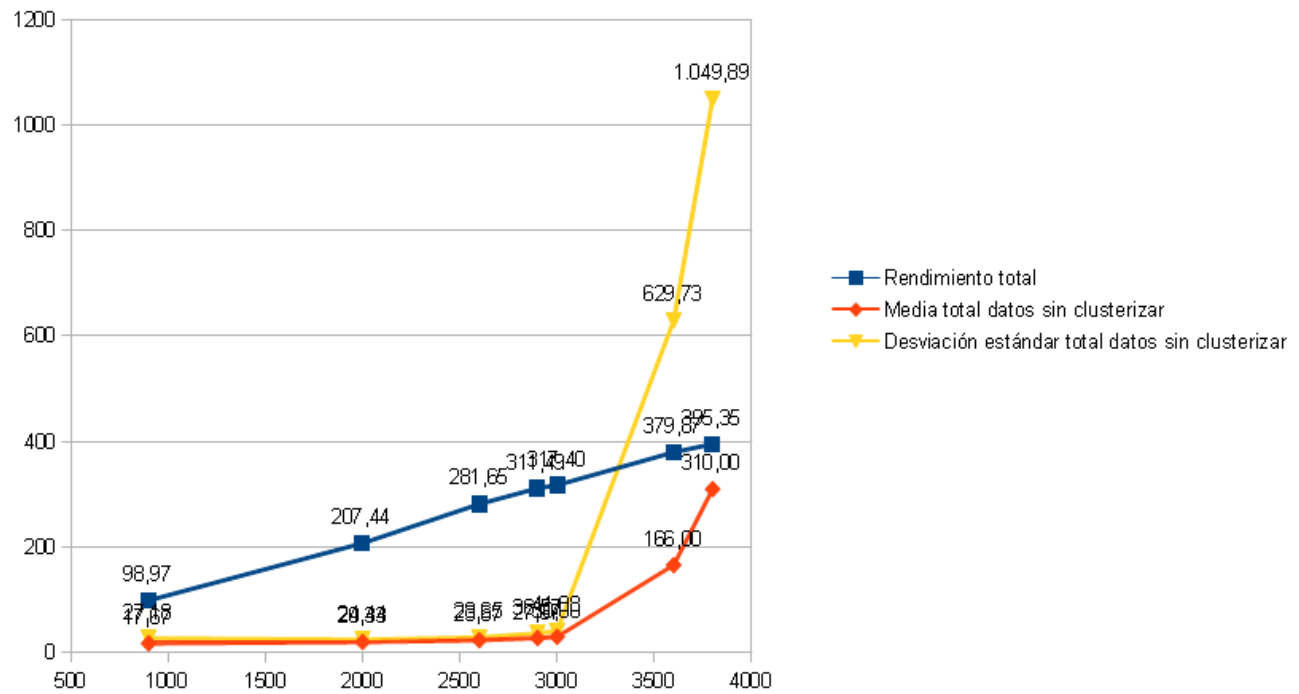
Sync Test - Individual



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

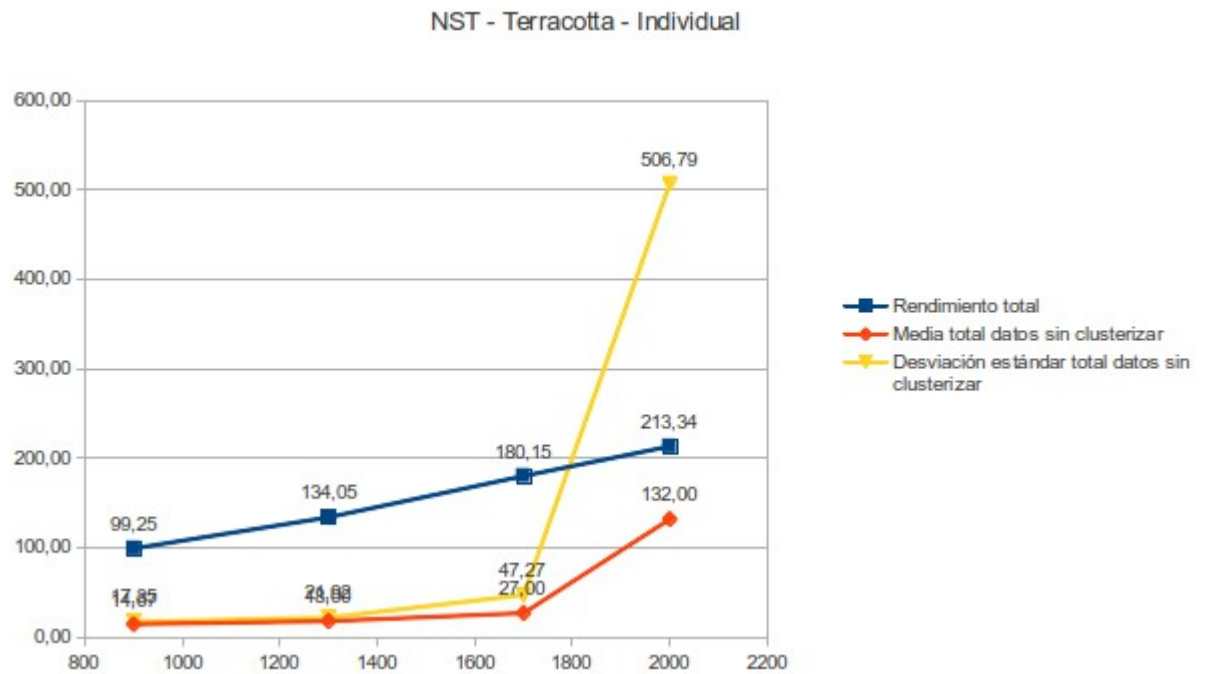
No Sync Test - Cluster

NST - Terracotta - Cluster



Se retorna a los patrones de comportamiento más frecuentes

No Sync Test - Individual



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.

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 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: observando la performance y rendimiento 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 así obtener una mayor tolerancia al incremento en la carga.

7. 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 25 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, JbuilderX, 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) con su implementación de Credenciales de identificación del personal, el sistema de Datawarehouse, el naciente sistema CONICET Digital, 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.

Dentro del grupo a modo global, actualmente me encuentro dirigiendo un subgrupo enfocado principalmente en el proyecto SIGERH, y en segundo término aunque adquiriendo cada vez más relevancia, en una próxima implantación de SIGEVA.

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: Ciudad de la Paz 3272 – 4/A, CP 1429, Ciudad Autónoma de Buenos Aires.
- Teléfono particular: 011 – 5339 – 8528.
- 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)

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.

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 Estadistica 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)

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

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

Taller De Programacion 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). Cátedra: Papaleo – Noremberg. (22/12/2009 L149 F120).
Administración y Control de Proyectos Informáticos I (código 7544).
Cátedra: Martínez. Nota: Final Pendiente (28/09/2009 L102 F45).
Calidad en el Desarrollo de Sistemas (código 7548). Cátedra: Pantaleo
Sistemas de Programacion no Convencional de Robots (código 7570).
Cátedra: Merlino
Manufactura Integrada por Computador (CIM) I (código 7565). Cátedra:
Ierache - Ochoa
Sistemas Automáticos de Diagnóstico y Detección de Fallas I (código 7567).
Cátedra: Merlino
Taller de Desarrollo de Proyectos II (código 7547). Cátedra: Fontela
Manufactura Integrada por Computadora (CIM) II (código 7566). Cátedra:
Ierache - Ochoa
Sistemas Automáticos de Diagnóstico y Detección de Fallas II (código 7569)
Administración y Control de Proyectos Informáticos II (código 7546).
Cátedra: Martínez
Lenguajes de Programación (código 7516)

Estadísticas a la fecha:

CBC: 6 materias

Carrera 10 – Ingeniería en Informática: 40 materias, Promedio 7,5

Suma de créditos: 236

8. 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>

[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>

[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>

[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

[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, Aref. 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. *How Terracotta Works* [en línea]. [San Francisco, California, Estados Unidos]. 2010 [ref. de julio 2011 a julio de 2012]. Disponible en Internet para usuarios registrados:

<<http://www.terracotta.org/platform/how-terracotta-works>>