

Speech

Presentación:

Editar: http://prezi.com/_rfr8obszr/edit/?auth_key=slxaqri&follow=xxeorf4i7_hd
(dbarrabino)

Ver: http://prezi.com/_rfr8obszr/clustering/

Minuta de la reunión del día sábado 27 de octubre:

1 Datos Cluster (resol comp) → En misma maq
- Dar baja 1 nodo / Ver cambia el sess ID
- Sincronizar
- Subir Nodo → (entrar al sist x ese nodo
con la consola de) baw.
- Sin
- bajar el otro
- Sin
- alta
- Sinc
Disp +
Replicas

2 **Cache** explicar que en Sbars
no se ve
→ Mostrar consola de terracotta
Fell
mostrar hist browser

01.- Datos cluster (resolución compartida) => En misma máquina: 1 en Firefox, 1 en Chromium

- => Primero ver qué nodo responde
- => Dar de baja un nodo ; verificar que cambia el session id
- => Sincronizar
- => Subir nodo ; entrar al sistema por ese nodo así el Balancer Manager le cambia el status a Ok
- => Sincronizar
- => Bajar el otro nodo => Disponibilidad + Replicación de sesión
- => Sincronizar
- => Alta
- => Sincronizar

02.- Cache

- Explicar que en JBoss no se ve
- Mostrar consola de Terracotta

03.- Antes la primera corrida: eliminar el histórico de los browsers

Antes de la corrida de JBoss: eliminar el histórico de los browsers

Minuta de la reunión del día sábado 31 de octubre:

- ¿Explicar el origen del nombre?

http://en.wikipedia.org/wiki/Terracotta_Army

The Terracotta Army or the "Terra Cotta Warriors and Horses", is a collection of terracotta sculptures depicting the armies of Qin Shi Huang, the first Emperor of China. It is a form of funerary art buried with the emperor in 210–209 BC and whose purpose was to protect the emperor in his afterlife.

- La primera parte de la presentación es de Martín (hasta "Entonces qué hacemos"), luego sigue Diego, luego mitad cada uno; la demo, vemos: y la parte de las conclusiones es mitad cada uno, Diego las dos primeras, Martín las dos segundas, y a partir de ahí lo que se pueda sumar, copado; en la parte de las conclusiones de alta performance, decir que tuvimos JMeter que es pesadísimo, corriendo en la misma PC
- Hay que repasar los puntos de la minuta del sábado 27/10/2012, ahí había anotaciones sobre la demo en sí, tal vez se pueda "guionar" la demo
- Inconvenientes encontrados (Diego): no hablar en la parte de "Cache", sobre PojoCache, es más para hablar de cache de persistencia, política de Infinispan, que hay que elegir qué se cachea, esas cosas; luego en la parte de "Replicación de datos", ahí si PojoCache; ir más a lo conceptual, no perder tiempo en detalles
- Parte en que se explican las implementaciones JBoss y Terracotta (Diego): hacer un corte, que se note que JBoss es una cosa, y que Terracotta + Tomcat es otra cosa; en la parte en de Terracotta, recordar decir que sirve para cualquier aplicación (salvo por retoques menores en el código)

- Ver cuál es la mejor manera de explicar que ambas implementaciones cumplen con lo que queremos
- Conclusiones: luego de la demo; hay que cambiar el flujo de la presentación
- Resultados de las pruebas: Diego; explicar la cantidad de pruebas que hicimos, que cada vez que descubríamos algo hacíamos todo nuevamente, la cantidad de gráficos que hicimos, cómo se interpretan los gráficos, qué hay en cada eje, etc
- Llevar cartelitos con nombres de las PCs ("worker1", "worker2", "basededatos / apache / terracotta")
- Llevar cable HDMI; quien sabe...
- Viernes, 19 hs, ¿en dónde?

Limitaciones:

- 45 minutos disponibles
- Imaginemos 15 minutos mostrando la demo
- Quedan 30 minutos para diapositivas

Estructura de la presentación (datos actualizados a las 17:31 del 02 de noviembre de 2012):

- Diapositiva 01: Presentación / Introducción (**Martín**)
- Diapositiva 02: Problema (**Martín**)
- Diapositiva 03: Necesitamos (**Martín**)
- Diapositiva 04: Aplicación de prueba (**Martín**)
- Diapositiva 05: Patrones de arquitectura y diseño (**Martín**)
- Diapositiva 06: Qué hacemos (**Martín**) => 10 minutos
- Diapositiva 07: Solución: Cluster (**Diego**)
- Diapositiva 08: Solución: Cluster (2) (**Diego**)
- Diapositiva 09: Funcionalidades necesarias del Cluster (**Diego**)
- Diapositiva 10: JBoss (**Diego**)
- Diapositiva 11: Terracotta + Tomcat (**Diego**)
- Diapositiva 12: Inconvenientes encontrados (**Diego**) => Otros 10 minutos, 20 en total
- Diapositiva 13: Pruebas - Manuales (**Martín**)
- Diapositiva 14: Pruebas - Automáticas (**Martín**)
- Diapositiva 15: Pruebas - Planillas (**Martín**) => 5 minutos, 25 en total
- Diapositiva 16: Pruebas - Resultados (**Diego**)
- Diapositiva 17: Comparativa Tiempos de Respuesta (**Diego**)
- Diapositiva 18: Comparativa Rendimientos (**Diego**) => 5 minutos, 30 en total
- Diapositiva 19: Cierre y paso a la demo (**Martín / Diego**) => Vemos
- Diapositiva 19: Conclusiones (**Martín / Diego**) => Vemos
- Diapositiva 21: Preguntas (**Martín / Diego**) => Vemos

Tiempo por diapositiva (datos actualizados a las 17:31 del 02 de noviembre de 2012):

- 17 diapositivas con contenido (la presentación y la de preguntas no cuentan)
- 30 minutos disponibles / 18 diapositivas con contenido = 1 minuto 40 segundos más o menos por diapositiva
- Aparentemente, lo recomendado es enunciar apróx. 150 - 170 - 200 palabras por minuto

Sóamente el texto de la presentación (datos actualizados a las 17:31 del 02 de noviembre de 2012):

- Diapositiva 01: Presentación / Introducción (**Martín**)

Qin - Cluster

Autores:

Barrabino, Diego

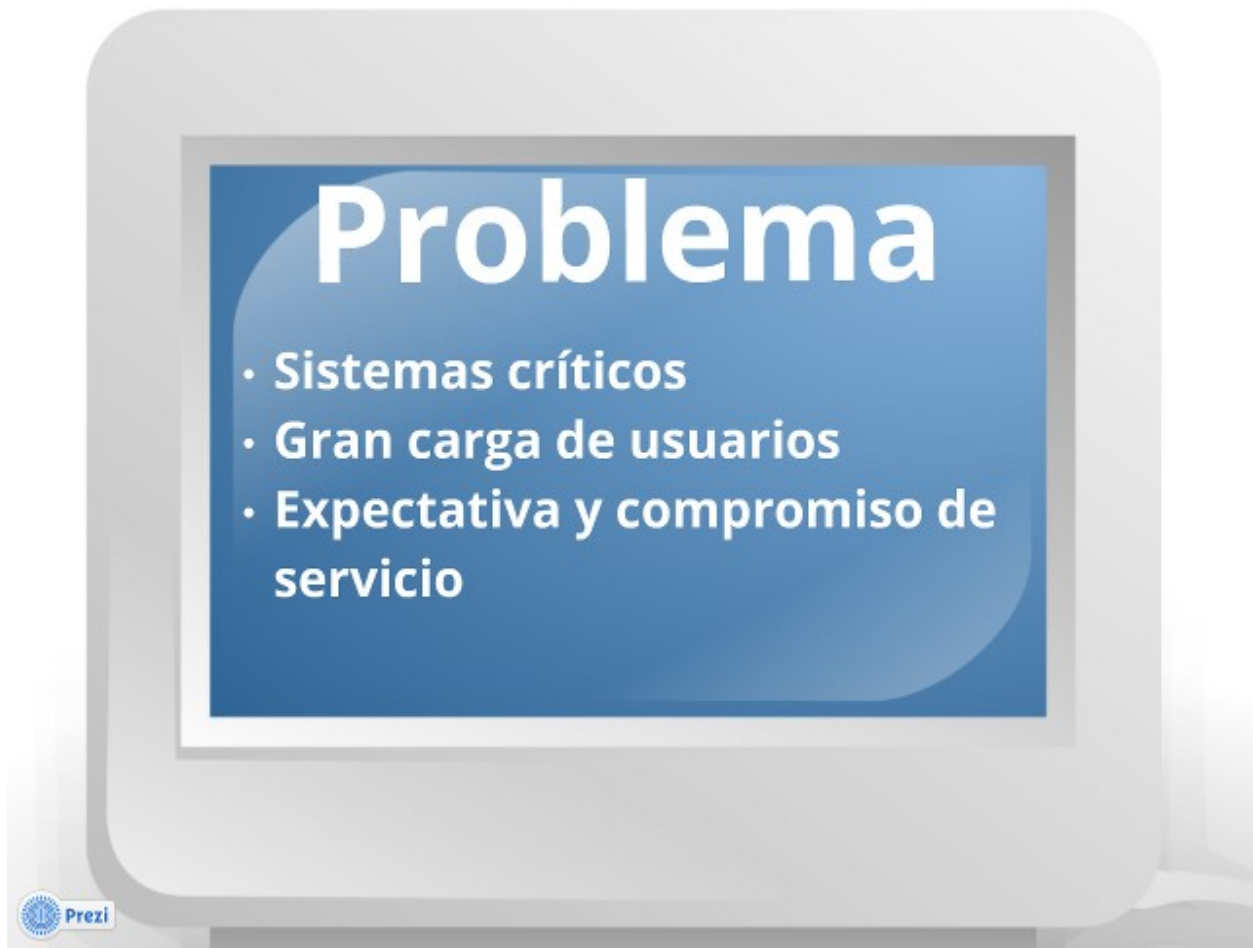
Moreyra, Martín Jorge



- Decir quienes somos !!!

Vamos a presentar el trabajo profesional titulado Qin - Cluster, que consiste principalmente en una comparación de dos arquitecturas para implementar cluster desde diferentes puntos de vista: tanto desde el punto de vista de arquitectura de software, las herramientas y sus resultados.

- Diapositiva 02: Problema (**Martín**)

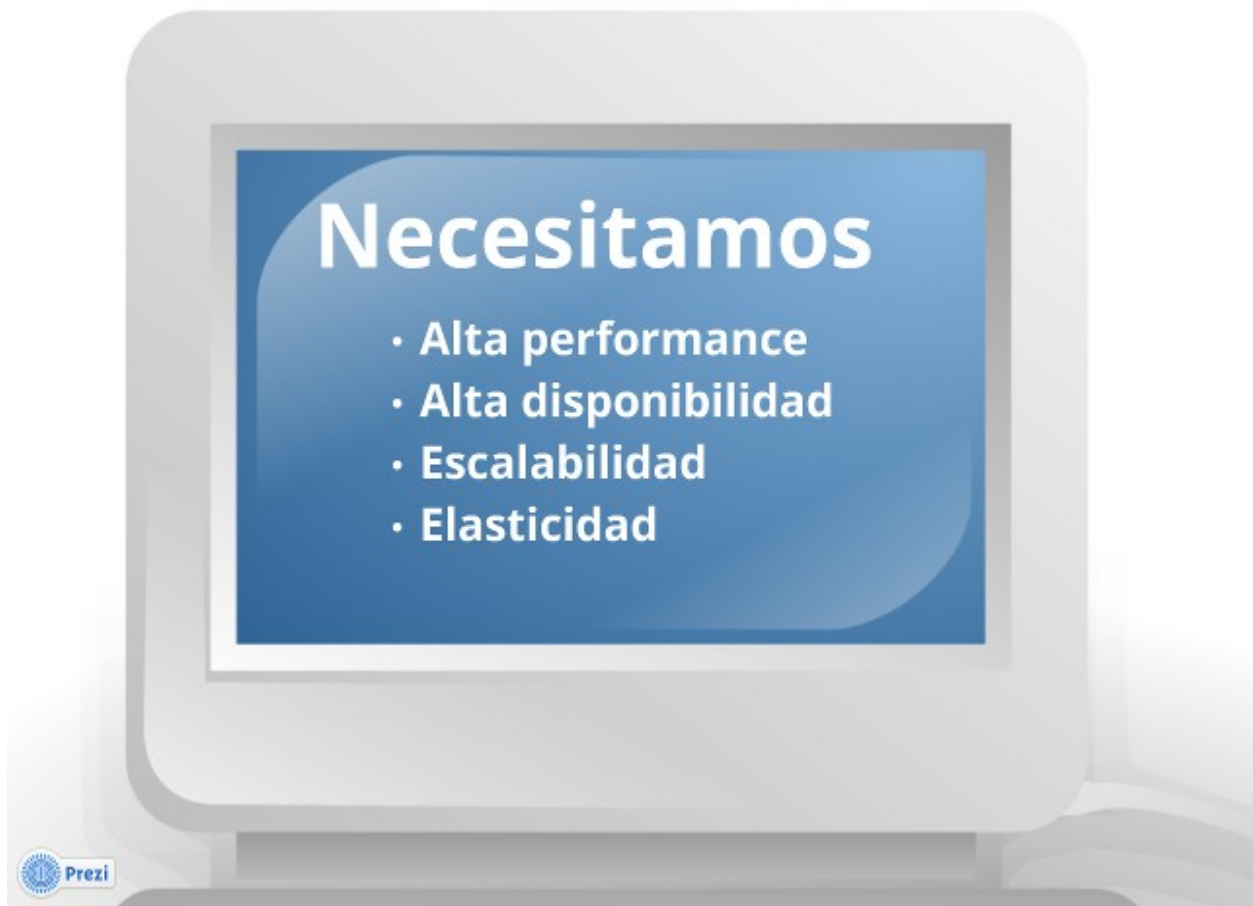


Hoy en día los sistemas informáticos se volvieron un elemento fundamental en infinidad de actividades humanas.

Existen diferentes problemas, asociados con diferentes tipos de sistemas:

Sistemas críticos	=> sistemas control de vuelos
Gran carga de usuarios	=> Redes sociales
Expectativa y compromiso de servicio	=> Bancos - uptime y velocidad

- Diapositiva 03: Necesitamos (**Martín**)



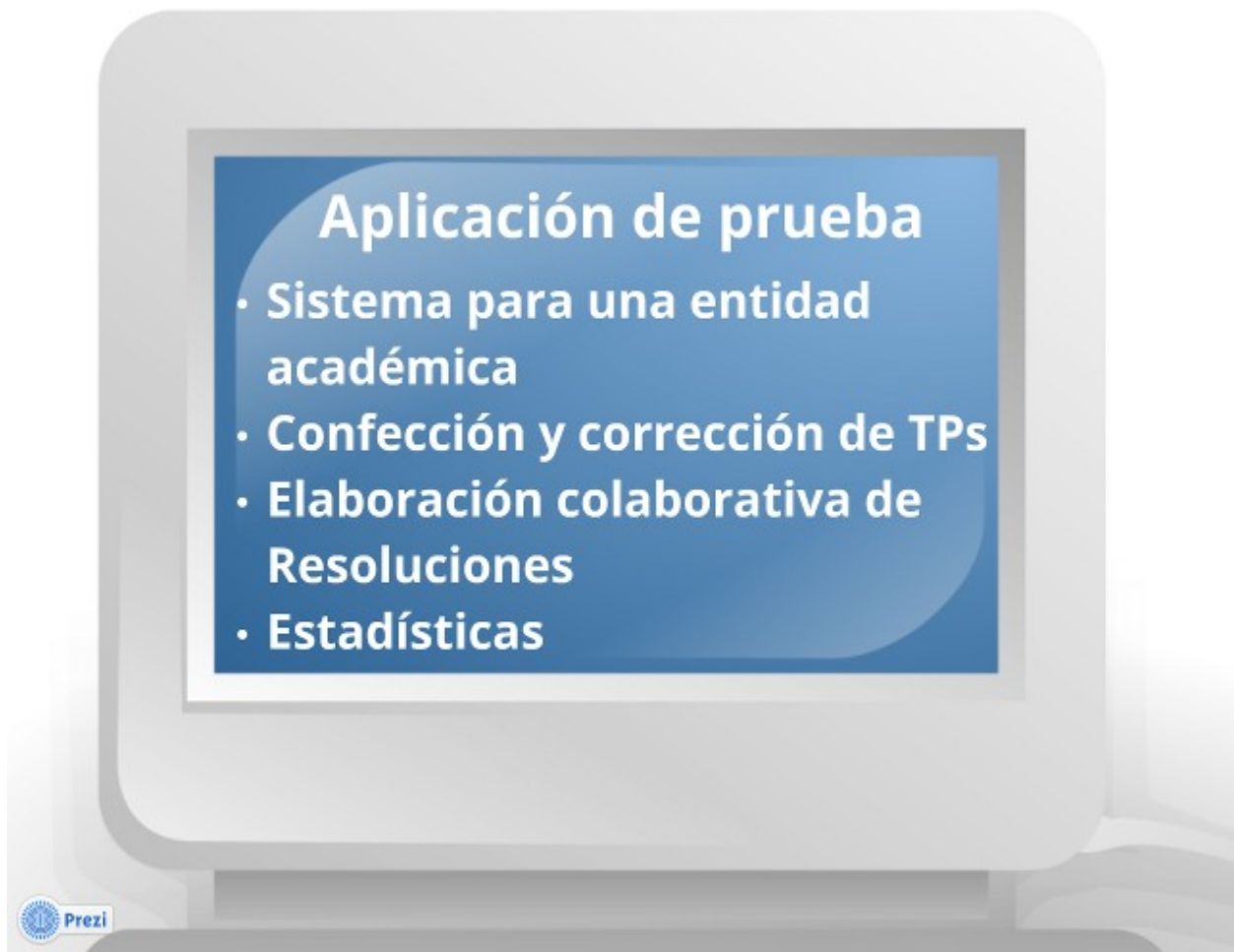
Alta performance: velocidad de procesamiento sist, latencia, tiempo de rta y rendim

Alta disponibilidad: grado de continuidad operacional

Escalabilidad: gran número de componentes (más carga, más componentes q la manejen). Vertical/horizontal

Elasticidad: facilidad y velocidad con la que se puede escalar la aplicación

- Diapositiva 04: Aplicación de prueba (**Martín**)



- Diapositiva 05: Patrones de arquitectura y diseño (**Martín**)

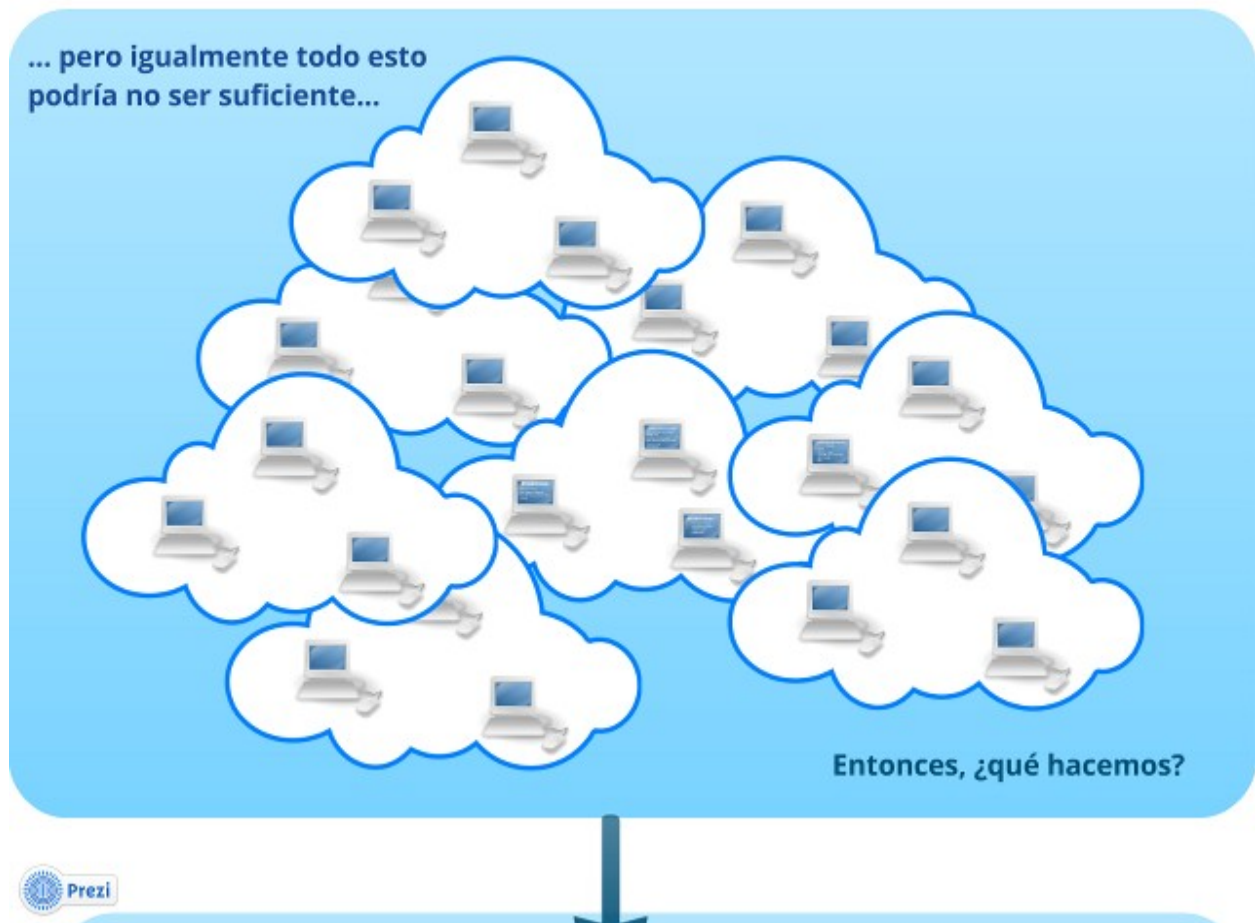


Arquitectura: config de elementos arquitectónicos y sus restricciones: elegir las restricciones para obtener las características. se refiere a estructura fundamental

Diseño: nivel más bajo, referidos a problemas de código de ciertos sectores

Layers / Tiers	: física y lógica
Domain model	=> Objetos : usamos objetos en todas las capas, no usamos DTO
Data mapper	=> Hibernate
MVC	=> Spring MVC
EAO	=> en el segundo gráfico

- Diapositiva 06: Qué hacemos (**Martín**)

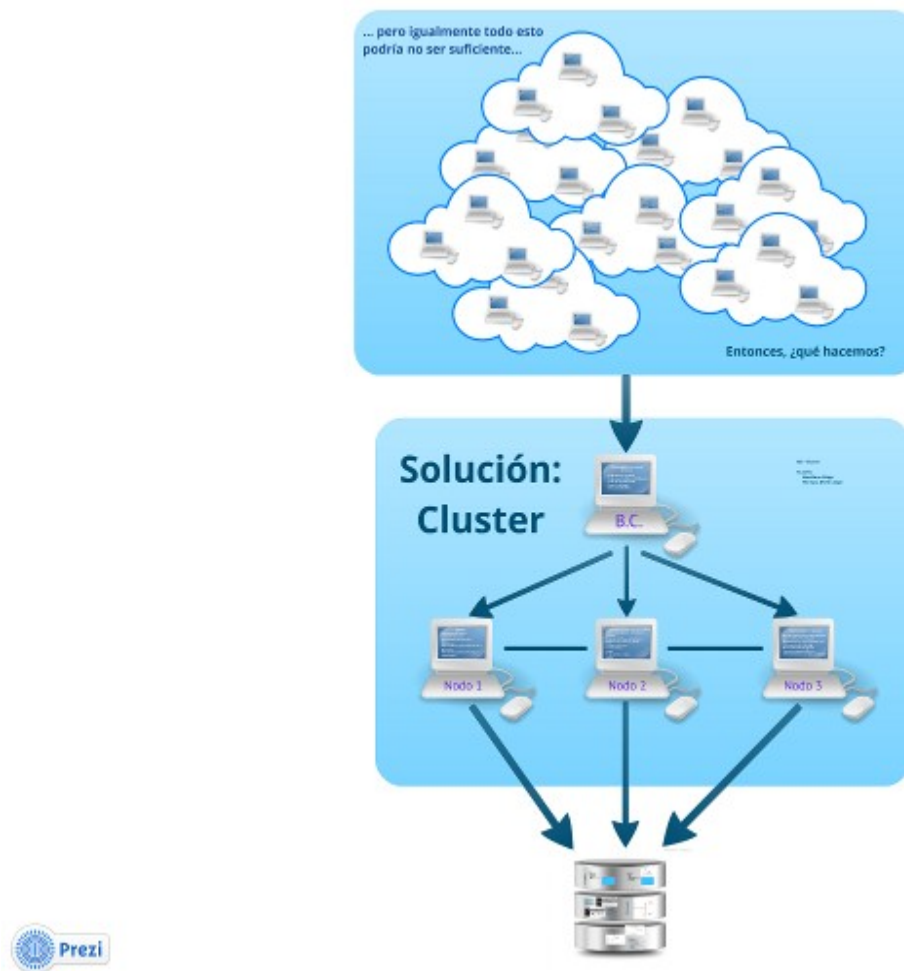


(**Martín**) problemas ya mencionados + problemas de hard + de conectividad etc

Lo común es organizar todo en un solo servidor; eso en algún momento va a colapsar; al igual que como ocurre con los discos rígidos, no importa lo grande que sea, va a colapsar; cuando eso pase, no se estará resolviendo el problema que presentamos, independientemente de aplicar todo lo que ya mencionamos que se debe aplicar

Se debe incorporar algo más

- Diapositiva 07: Solución: Cluster (**Diego**)



(nubes de usuarios + bc + nodos + db)

El cluster !

- Qué es un cluster

- Cluster homogéneo / heterogéneo, horizontal / vertical ?

- Dejamos afuera el estudio de la base de datos; de última se pone más plata

Para superar las limitaciones mencionadas, proponemos armar un Cluster, que es el objeto de nuestra investigación

Un Cluster es un conjunto de servidores llamados nodos, que se comporta como un único servidor

Esta técnica es recomendada para actividades de supercómputo, software de misiones críticas, comercio electrónico, bases de datos de alto rendimiento, etc.; la importancia de las áreas en que se aplica, nos motivó a realizar esta investigación

Existen distintas maneras de clasificar Clusters

Según hardware y software:

- Cluster homogéneo: los nodos tienen la misma configuración de hardware y software
- Cluster semi-homogéneo: cada nodo tiene diferente rendimiento, pero con hardware y software similares

- Cluster heterogéneo: cada nodo tiene diferente hardware y/o software

Según topología:

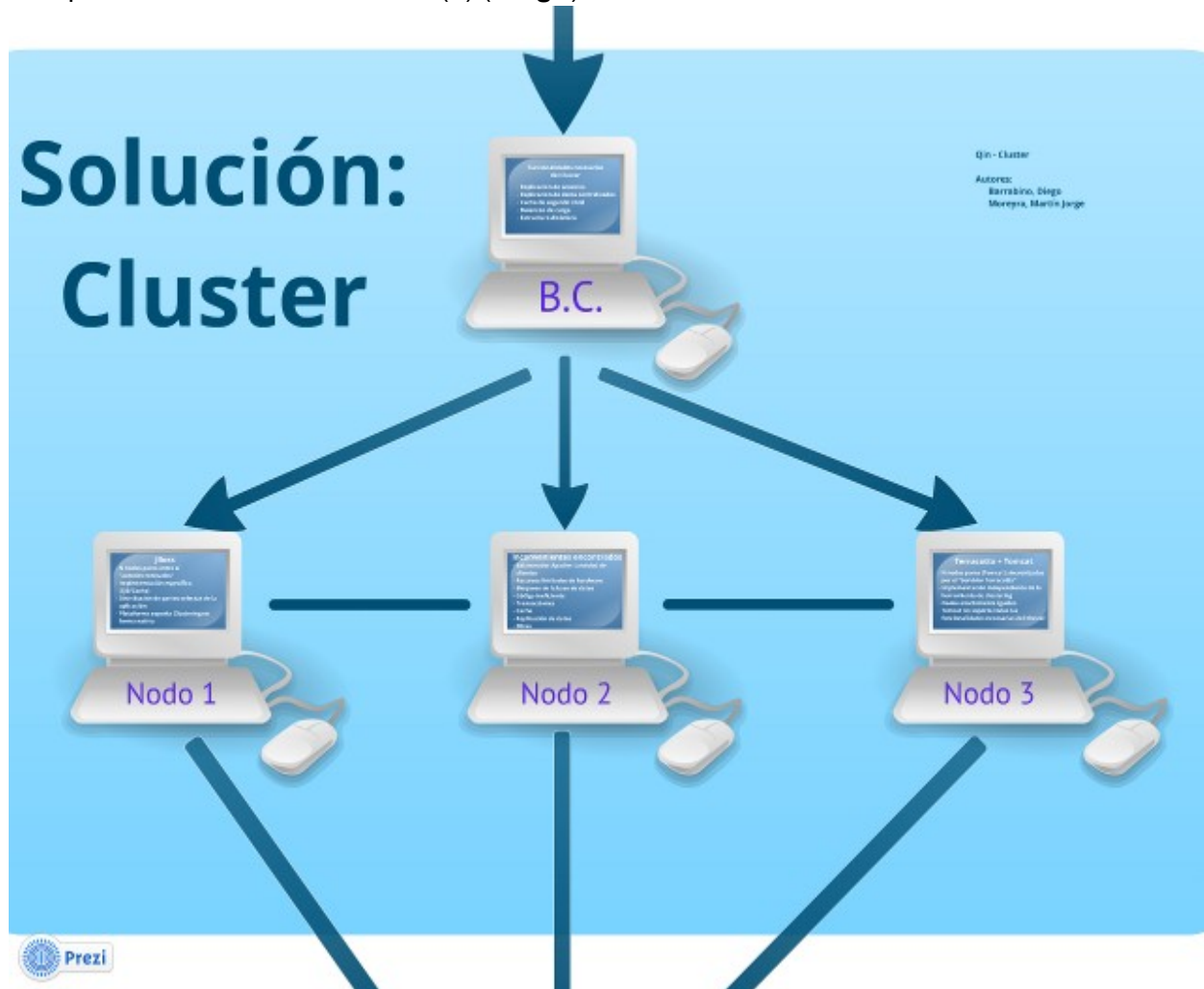
- Cluster horizontal: a cada nodo le corresponde una computadora => ideal para cuando se dispone de servidores corrientes

- Cluster vertical: los nodos residen en la misma computadora => ideal para sacar provecho de servidores potenciados

- Combinación de lo anterior

Para que un Cluster funcione como tal, es necesario contar con un sistema de administración, el cual se encargue de interactuar con los procesos que corren en él para optimizar el funcionamiento.

- Diapositiva 08: Solución: Cluster (2) (Diego)



(bc + nodos)

- Cluster

- Hablar sobre implementación particular para nuestro TP
- Funcionalidad para sincronizar para hacer las cosas algo más complicadas
- JBoss

Algo más a priori

- Terracotta
- Generalidades

Focalizando en nuestro trabajo, cabe destacar que:

- No hemos profundizado en el servidor de base de datos
- Siempre hemos trabajado con clusters homogéneos y horizontales
- Del menú de herramientas disponibles en mercado que permiten armar y sincronizar clusters, elegimos las dos siguientes, debido a que pudimos comprobar que son muy utilizadas y además tienen características distintas, lo que enriquece la investigación:
 - JBoss: servidor web enterprise; permite replicar y sincronizar por sí mismo, partes de una aplicación entre varias instancias

- Al desarrollar una aplicación específicamente pensada para funcionar con esta herramienta y usufructuar sus servicios especializados nativos, se obtiene un muy buen rendimiento.

- Lo anterior condiciona el diseño de la aplicación, para mal o para bien

- Terracotta: librería que posibilita escalar aplicaciones Java; permite sincronizar externamente N nodos que deben ser funcionales en sí mismos, por lo cual en este caso lo combinamos con Tomcat

- No se requieren implementaciones específicas, por lo que en teoría se puede replicar cualquier aplicación, aunque el rendimiento podría no ser el ideal en pos de la genericidad.

Este es el esquema ideal que proponemos: un cluster compuesto de tres nodos y un servidor de base de datos

Funcionalidades necesarias del Cluster

- Replicación de sesiones
- Replicación de datos centralizados
- Cache de segundo nivel
- Balanceo de carga
- Estructura dinámica



Replicación de sesiones
Replicación de datos centralizados
Cache de segundo nivel
Balanceo de carga
Estructura dinámica

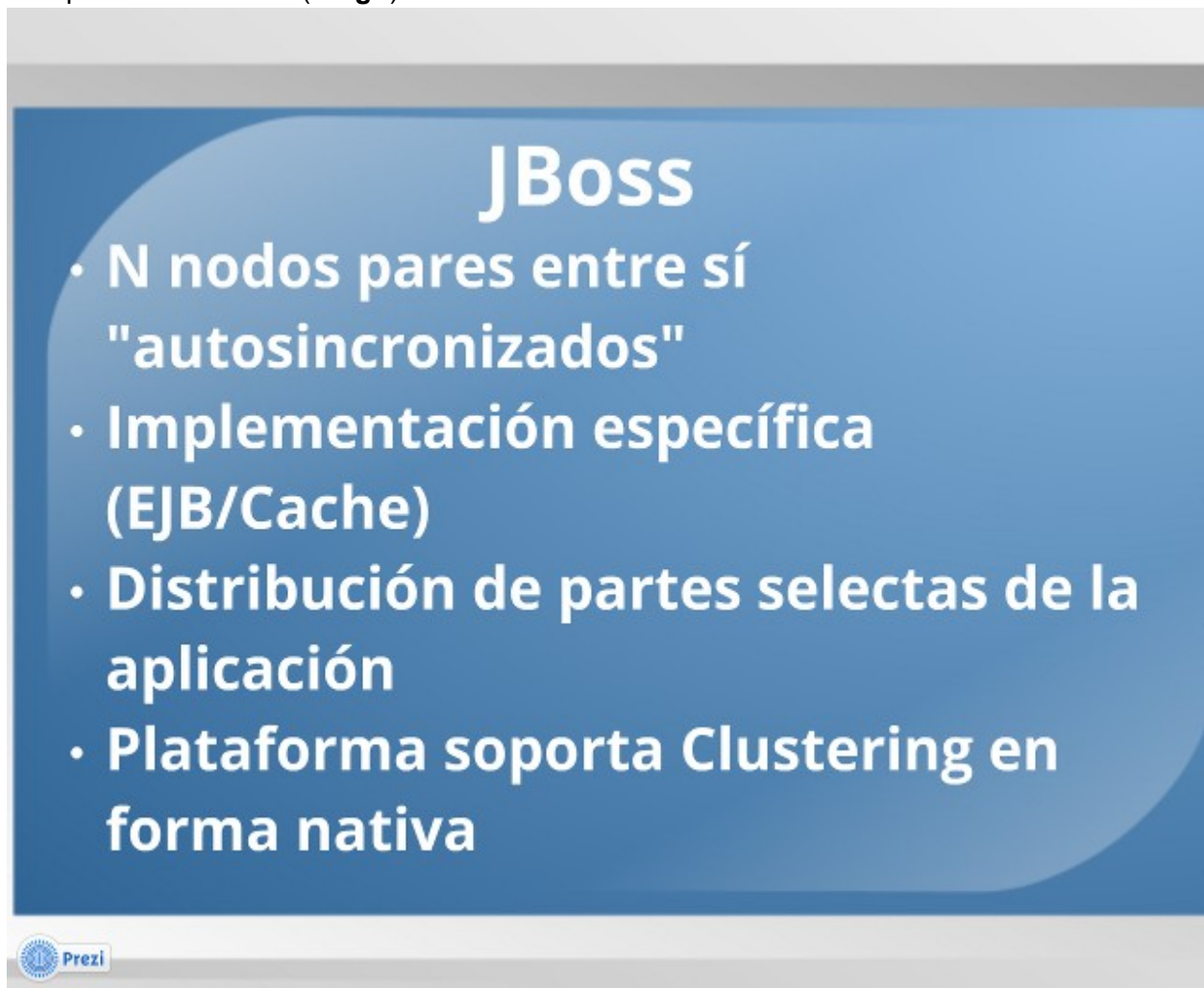
Consideramos que hay cinco aspectos de un cluster, que son los que a nuestro criterio justifican la decisión de utilizarlo:

- Replicación de sesiones: a partir de que se provee servicio a algún usuario desde alguno de los nodos, dicha sesión debe estar disponible para todos los demás nodos, de modo tal de compensar la falla de alguno de ellos
- Replicación de datos centralizados: se debe poder implementar un servicio de datos centralizados a nivel cluster, al que puedan acceder todos los nodos irrestrictamente
- Cache de segundo nivel: se debe poder implementar a nivel cluster, un cache de segundo nivel de persistencia, para optimizar el tiempo de respuesta de las consultas de la base de datos que hacen los distintos nodos
- Balanceo de carga: si bien esto es externo al cluster, consideramos pertinente a nuestra investigación, incorporar un balanceador de carga para este trabajo, para entre otras cosas

interactuar con el cluster mediante un sólo IP, el del balanceador; utilizamos Apache y su balanceador de carga mod_proxy

- Estructura dinámica: se debe poder agregar o retirar nodos a voluntad

En cada una de las implementaciones que hicimos, una para JBoss y otra para Terracotta, se alcanzaron estos cinco objetivos

A presentation slide with a blue background and a white curved shape on the left. The title 'JBoss' is in large white font. Below it are four bullet points in white text. At the bottom left is a small 'Prezi' logo.

JBoss

- N nodos pares entre sí "autosincronizados"
- Implementación específica (EJB/Cache)
- Distribución de partes selectas de la aplicación
- Plataforma soporta Clustering en forma nativa

Prezi

N nodos pares entre sí "autosincronizados"
Implementación específica (EJB/Cache)
Distribución de partes selectas de la aplicación
Plataforma soporta clustering de forma nativa

- Explicar punto por punto

Aspectos particulares de la implementación con JBoss:

- N nodos pares entre sí autosincronizados: Al levantar los nodos que forman el cluster, éstos automáticamente se estructuran como tal y se sincronizan automáticamente; el que levanta por completo antes que los demás, oficia de nodo principal, y es donde se instancian los servicios únicos a nivel cluster
- Implementación específica (EJB/Cache): la aplicación desplegada en JBoss es la misma que la desplegada en Terracotta + Tomcat, con la única diferencia de que la configuración del cache de segundo nivel, la replicación de datos centralizados y la configuración de la distribución de las sesiones de usuario son específicas para JBoss. También se podría haber utilizado EJB lo cual podría haber mejorado la calidad de la solución, pero se habría

complicado la comparación con Terracotta con el que se utiliza Tomcat que no soporta EJB, por lo cual se desestimó.

- Distribución de partes selectas de la aplicación: se puede instanciar un único replicador de datos centralizados (en el nodo principal), y también se puede desplegar todas las demás funcionalidades en todos los nodos; se pueden hacer combinaciones que podrían ser útiles según el caso

- Soporte de clustering de forma nativa: con sólo levantar los nodos en cuestión, se arma un cluster, sin requerir agregado de componentes externos

Bien, estos han sido los detalles de una de las dos implementaciones de cluster que hicimos; pasamos a los detalles de la otra implementación

Terracotta + Tomcat

- N nodos pares (Tomcat) sincronizados por el "Servidor Terracotta"
- Implementación independiente de la herramienta de clustering
- Nodos exactamente iguales
- Tomcat no soporta todas las funcionalidades necesarias del Cluster



N nodos pares (Tomcat) sincronizados por el "Servidor Terracotta"

Implementación independiente de la herramienta de clustering

Nodos exactamente iguales

Tomcat no soporta todas las funcionalidades necesarias del Cluster

Aspectos particulares de la implementación con Terracotta + Tomcat:

- N nodos pares (Tomcat) sincronizados por el "Servidor Terracotta": los nodos del cluster de Terracotta implementados con Tomcat, por sí solos no se identifican. Es Terracotta el que los administra y sincroniza, armando de esta manera el cluster. Además, Terracotta también ofrece funcionalidades centralizadas a nivel cluster, sincroniza las sesiones de usuario, sincroniza los distintos cache de segundo nivel haciéndolos trabajar como uno solo y permite sumar cuantos nodos se quiera; es el que aporta la inteligencia del cluster
- Implementación independiente de la herramienta de clustering: la implementación de los nodos en sí no tiene punto de contacto alguno con el armado del cluster, son dos funcionalidades distintas llevadas a cabo por dos herramientas distintas, por lo cual, en teoría, se puede aplicar Terracotta sobre cualquier aplicación
- Nodos exactamente iguales: los nodos son considerados iguales entre sí, a diferencia de JBoss en que el primero completamente levantado es el principal

- Tomcat no soporta todas las funcionalidades necesarias del Cluster: tal como se explicó, se depende de Terracotta, a diferencia del caso con JBoss

Inconvenientes encontrados

- Balanceador Apache: cantidad de clientes
- Recursos limitados de hardware
- Bloqueos de la base de datos
- Código ineficiente
- Transacciones
- Cache
- Replicación de datos
- Otros



Experimentando con ambas implementaciones, dimos con varios problemas que pudimos solucionar, de los que mencionaremos solo algunos:

Tuvimos que llevar al máximo posible, los parámetros de:

- Cantidad de conexiones de Apache
- Cantidad de conexiones del balanceador de carga
- Cantidad de conexiones del motor de base de datos
- Cantidad de archivos abiertos soportados por el SO
- Tamaño de buffers de comunicación del SO

... porque de no hacerlo así, las pruebas más exigentes no llegaban a realmente poner a prueba el cluster, ya que eran limitadas por los parámetros anteriores

Código ineficiente

Varios detalles del código fuente influían negativamente en los resultados, como ser nivel de log excesivo, pobre manejo de excepciones y algoritmos ineficientes. Incluso llegamos a construir un profiler casero para detectar los métodos que insumían mucho tiempo y poder mejorarlos

Transacciones

Inicialmente todos los servicios estaban marcados como transaccionales. Llegamos a ver que dichas transacciones se bloqueaban entre sí, por lo cual debimos configurarlas como de solo lectura y como adosables a otras transacciones en los métodos que efectuaban solamente consultas a la base de datos

Cache

Descubrimos que es vital determinar correctamente qué entidad se debe cachear y en dónde utilizar dichas entidades, ya que de no hacerlo se incurre en demoras. Por caso, tuvimos problemas al cachear datos "no muy estáticos" y tuvimos problema con las caches distribuidas en el caso de JBoss, ya que en vez de sincronizar, envía mensajes de total recambio de los datos

Replicación de datos

En el caso de JBoss, la versión usada no tiene integrado el componente adecuado para replicar datos, así como sí ocurre en justo la versión previa y a medias en la versión posterior, por lo cual tuvimos que incorporar el componente de la versión previa "a mano"

Otros

Velocidad de los núcleos del procesador

Servidores con modo debug desactivado

Primero elegimos el balanceador de carga mod_jk pero finalmente lo reemplazamos por mod_proxy

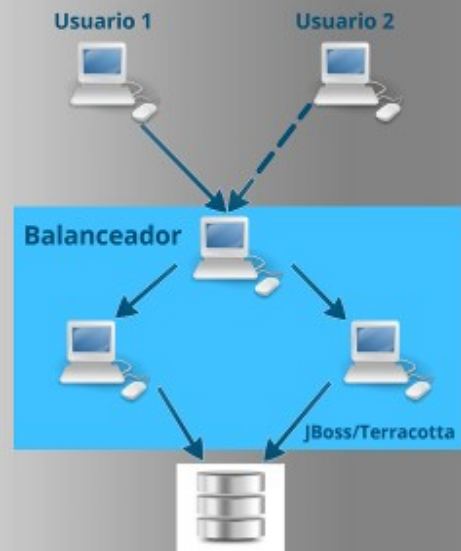
Muchas veces se corrompió la comunicación entre nodos de JBoss

Cada vez que identificamos y solucionamos un problema, volvimos a hacer todas las pruebas

Pruebas

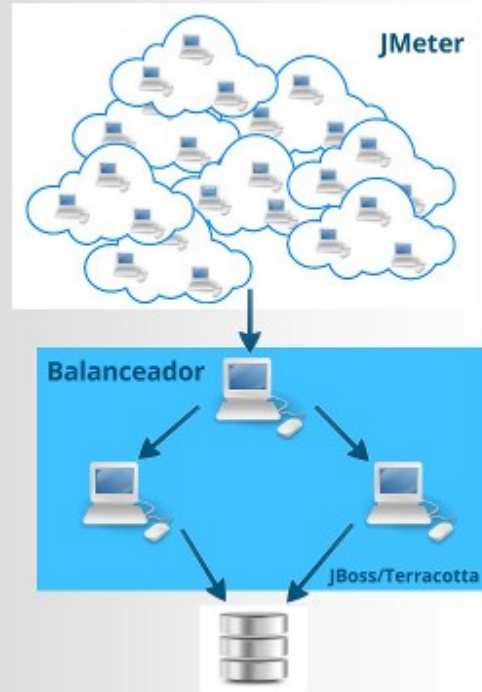
Manuales

- Disponibilidad - desconexión
- Replicación de sesion
- Cache

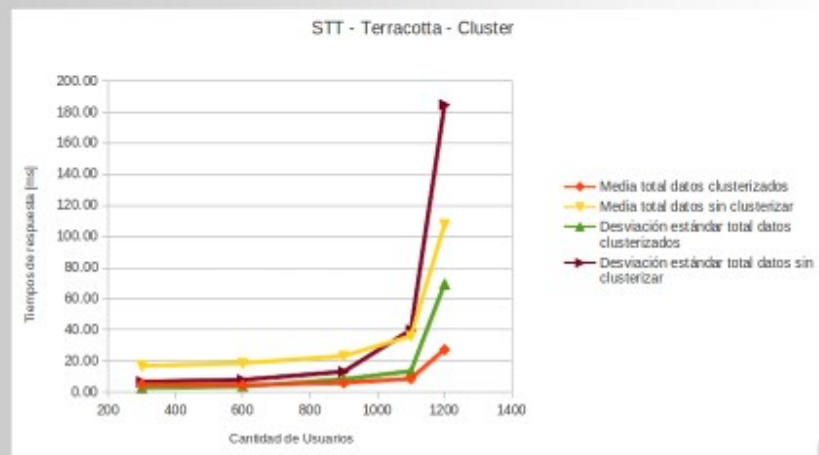
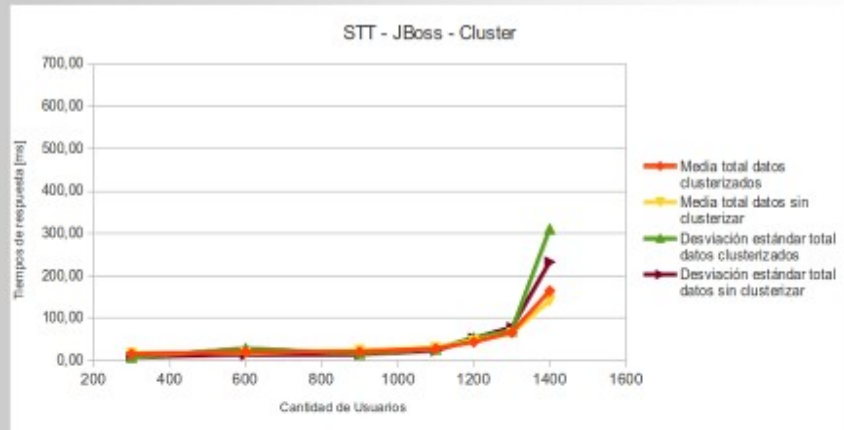


Automáticas

- Stress
- Carga
- Carga datos sincronizados
- Carga datos no sincronizados



Resultados



Definimos para cada una de las cuatro pruebas automáticas pasos de usuarios simulados y también los parámetros de tiempo promedio entre usuarios, tiempo de puesta en régimen, etc. Posteriormente, se ejecuta la prueba tanto para el cluster como para un nodo representativo individual, y se almacenan los datos en un planilla de cálculos preformateada. Posteriormente, concentramos todos los números en otra planilla de cálculos, y armamos los gráficos tanto de rendimiento como de tiempos

Los gráficos de rendimiento miden peticiones por segundo para cada paso de usuarios, y los gráficos de tiempos miden medias y desviaciones estándar de tiempos de servicio por cada paso de usuarios; la media es el promedio, y la desviación estándar mide cuánto se alejaron las mediciones de ese promedio, nos interesan las desviaciones estándar pequeñas, que representan un mejor servicio

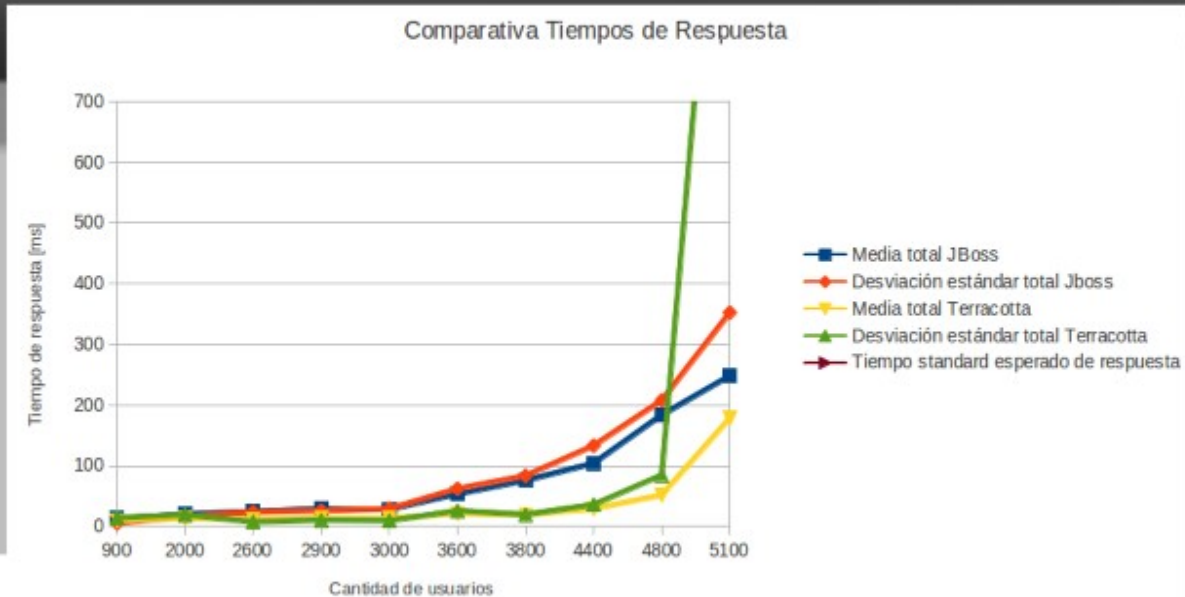
En este caso, estamos viendo los resultados de la prueba de stress para clusters

Cada vez que dimos con un inconveniente que incidía en los resultados y lo solucionamos, repetimos absolutamente todas las pruebas automáticas hechas hasta el momento para obtener mejores resultados

En total, considerando incluso las pruebas de cache y las pruebas con el profiler casero, y sin considerar todas las repeticiones por inconvenientes solucionados, hicimos 331 pasos de

pruebas automáticas oficiales en esta investigación; considerando las repeticiones, creemos haber hecho en total al menos 960 pasos de pruebas automáticas

- Diapositiva 16: Comparativa Tiempos de Respuesta (**Diego**)



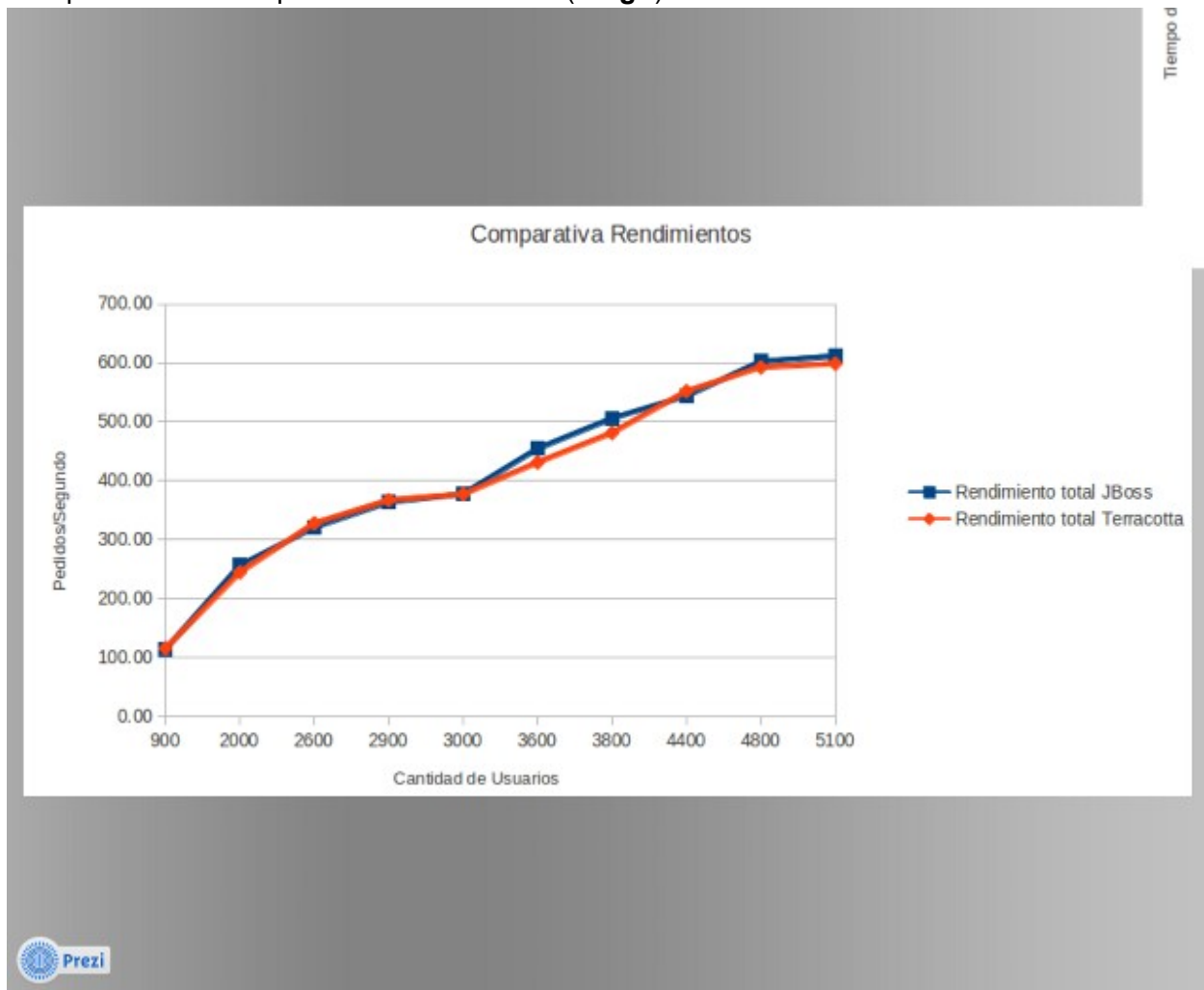
A partir de los gráficos de los resultados de las pruebas, hicimos comparativas entre todas las mediciones de los clusters y de los nodos individuales

Esta comparativa de tiempos es de la prueba de carga, que es la más representativa

Observando los tiempos de respuesta conseguidos en ambas implementaciones de cluster, podemos ver que JBoss es más predecible, ya que a medida que aumenta la carga, los tiempos de respuesta se van incrementando paulatinamente. Por el contrario, en el caso de Terracotta + Tomcat, vemos que al aumentar la carga sobre el sistema llegamos a un punto de quiebre, en donde los tiempos de respuesta se disparan. Podemos decir que la arquitectura implementada con JBoss es una arquitectura mas elástica que la implementada con Terracotta + Tomcat ya que al ser mas predecible se pueden tomar las acciones necesarias para aumentar la cantidad de nodos del Cluster y asi obtener una mayor tolerancia al incremento en la carga.

La tendencia se mantiene en las pruebas individuales; la única diferencia con los números de los clusters, es que con clusters se puede dar servicio a más usuarios, pero la velocidad request a request disminuye

- Diapositiva 17: Comparativa Rendimientos (**Diego**)

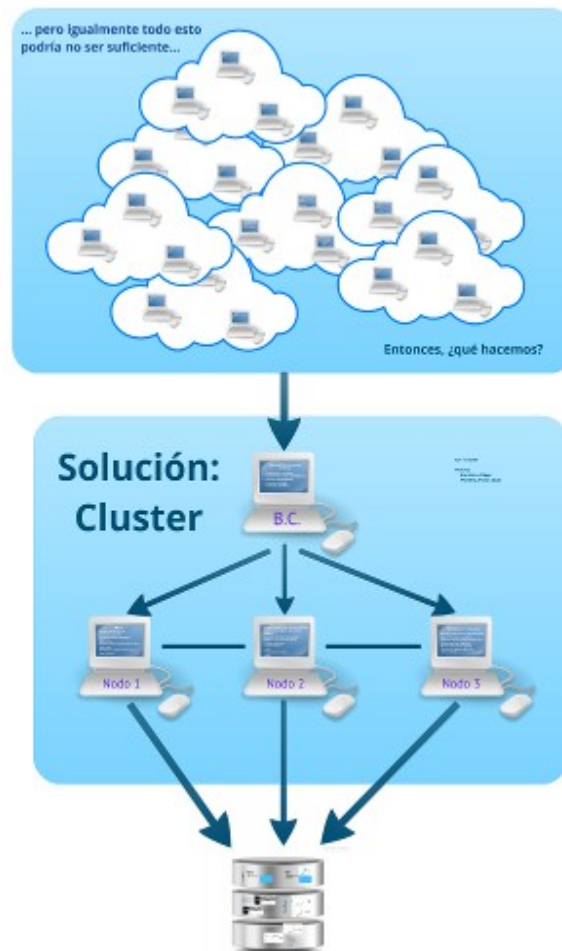


- Interpretación de los gráficos, que son de LT solamente
- Marcar detalles

Esta comparativa de rendimientos es de la prueba de carga, que es la más representativa. Sin embargo, en términos de rendimiento, ambas implementaciones básicamente dan los mismos resultados, tal vez con una leve diferencia en favor de JBoss. Esto no lo pudimos imaginar a priori, siempre nos manejamos con la sensación de que la implementación con JBoss debería prevalecer, considerando el nivel de sofisticación de la herramienta frente a Tomcat, por lo cual esto terminó siendo una muy agradable sorpresa. Consideramos que esta paridad se debe a que:

- No hay diferencias salientes a nivel código fuente de la aplicación de prueba para uno y otro cluster
 - Tomcat compensa con velocidad, lo que no ofrece en sofisticación, por eso es que llega al punto de quiebre del gráfico anterior básicamente sin preaviso
 - Terracotta realiza sus tareas de manera prácticamente ideal en este contexto
- La tendencia se mantiene en las pruebas individuales

- Diapositiva 18: Cierre y paso a la demo (**Martín / Diego**)



?

Bien, ahora pasamos a la demo

?

- Preparar worker1 para que presumiblemente atienda al usuario 1 y que sea el nodo principal
- Preparar dos navegadores y dos consolas
- En un navegador el usuario 1 contra el nodo que sea
- En otro navegador el usuario 2 contra el otro nodo
- Preparar dos consolas, una con el log de worker1 y otra con el log de worker2
- Mostrar en cada navegador los usuarios => Consulta
- Mostrar en cada navegador los workers => Home
- Que usuario 1 vaya a resolver el TP 1
- Que el usuario 2 vaya a sumarse a la resolución de TP 1 del usuario 1
- Sincronizar palabras
- Bajar worker1
- Mostrar que sigue habiendo sincronización de texto y que el usuario 1 sigue conectado (Consulta) pero ahora lo atiende otro worker

Balanceo de carga (inteligente): un usuario en cada nodo

Sincronización de datos centralizados: el texto que se sincroniza

Replicación de sesiones, sino el usuario 1 debería haber sido eliminado

Estructura dinámica, sino no habría sido tan fácil eliminar un nodo

(Cache: podríamos tener preparado de antemano la pantalla con el caché de JBoss, por más que sea espantoso)

- Levantar el nodo 1; esperar un tiempito prudencial

- Bajar nodo 2 => lo mismo

- Levantar el nodo 2; entrar con el usuario 3, 4, 5 hasta que se pueda volver a ver el nodo 2

Conclusiones

- **Alta performance**

4500 - 6000 usuarios en entorno de pruebas

- **Alta disponibilidad**

Redundancia datos y sesiones

- **Escalabilidad**

Herramientas que permiten agregar recursos

- **Elasticidad**

JBoss provee una arquitectura más elástica



Alta performance: con limitaciones de hardware y ejecutando en los nodos las pruebas con JMeter, que son muy demandantes de recursos, pudimos dar servicio sin problemas en nuestro sistema de prueba, a entre 4500 y 6000 usuarios; por lo cual, es válido suponer que aplicando estas técnicas en servidores dedicados y utilizando la infraestructura ideal recomendada, se podría dar un servicio de calidad a muchos más usuarios

Alta disponibilidad: demostramos que el cluster tiene la capacidad de prestar servicio incluso bajo condiciones problemáticas, y en gran parte eso se debe a la redundancia de datos y sesiones dentro del cluster, que posibilitan que si un nodo está caído, otro nodo pueda tomar su lugar al menos temporalmente

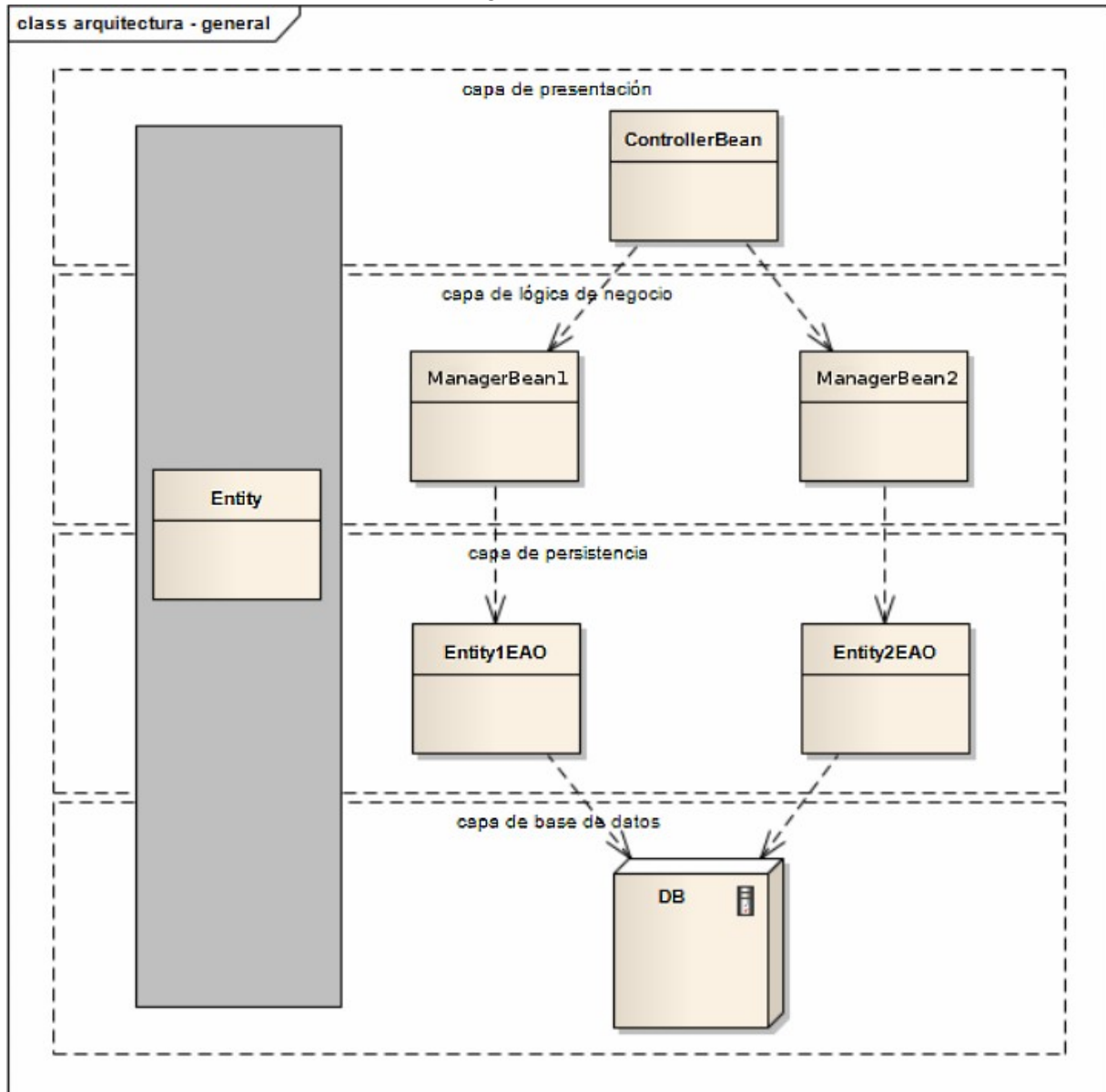
- Diapositiva 20: Preguntas (**Martín / Diego**)



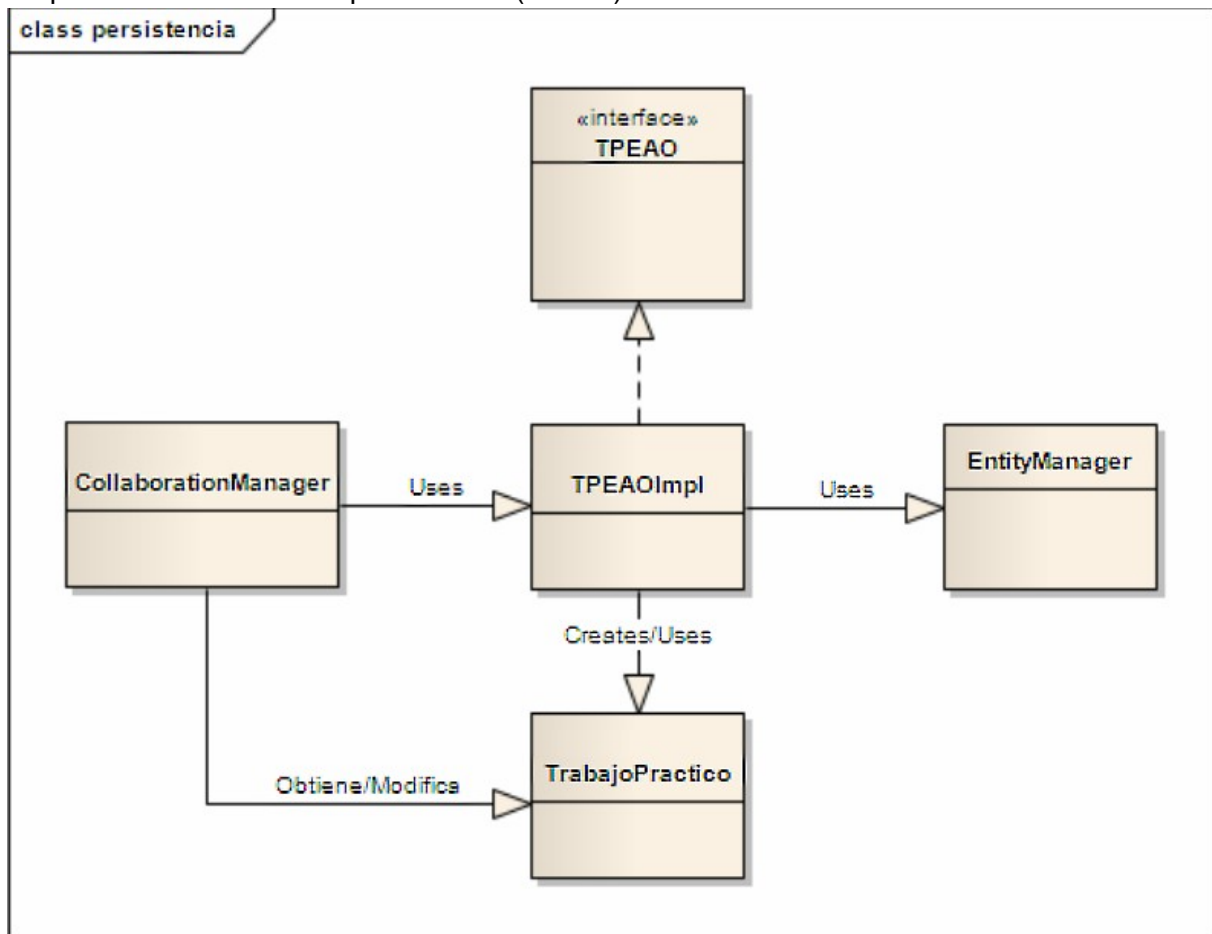
Preguntas ?

ANTERIORES

- Diapositiva 04: Gráfico de arquitectura general (**Martín**)



- Diapositiva 05: Gráfico de persistencia (**Martín**)



- Explicar los patrones, muy simple; tal vez si dá hablar de Managers, en cuanto a que son como una capa que entre otras cosas favorece a que no haya tantas llamadas a EJBs distintos, si hubiéramos usado EJB; tal vez dé para mencionar que inicialmente se pensó en usarlos

- Diapositiva 14: Gráficos de pruebas de clusters (**Martín**)

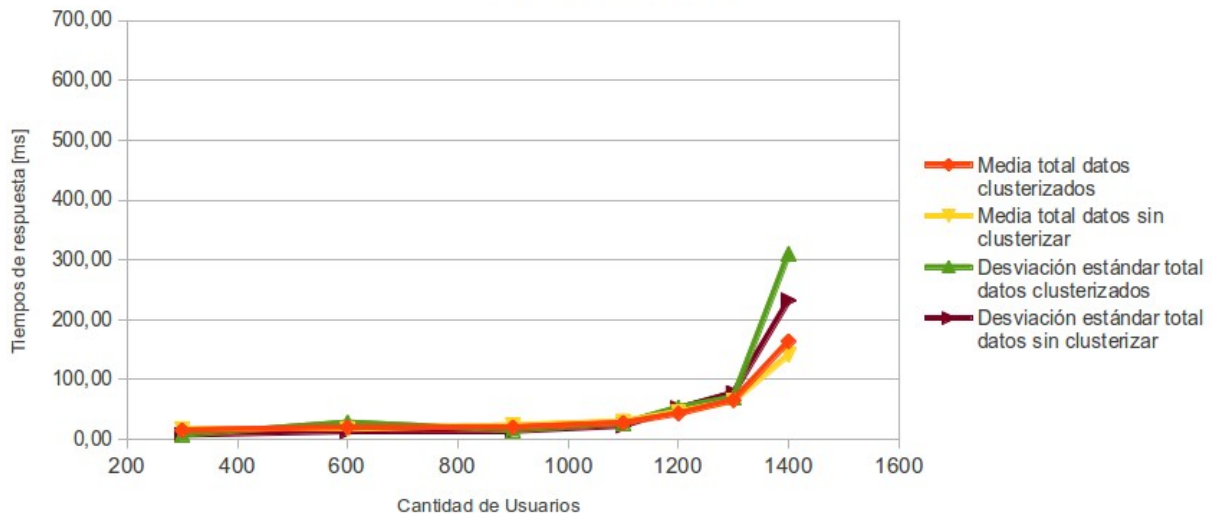
Mencionar tiempos standard de rta .1 ui, 1 naveg, 10 graficos y reportes

-solo mostramos STT pero hay muchos mas

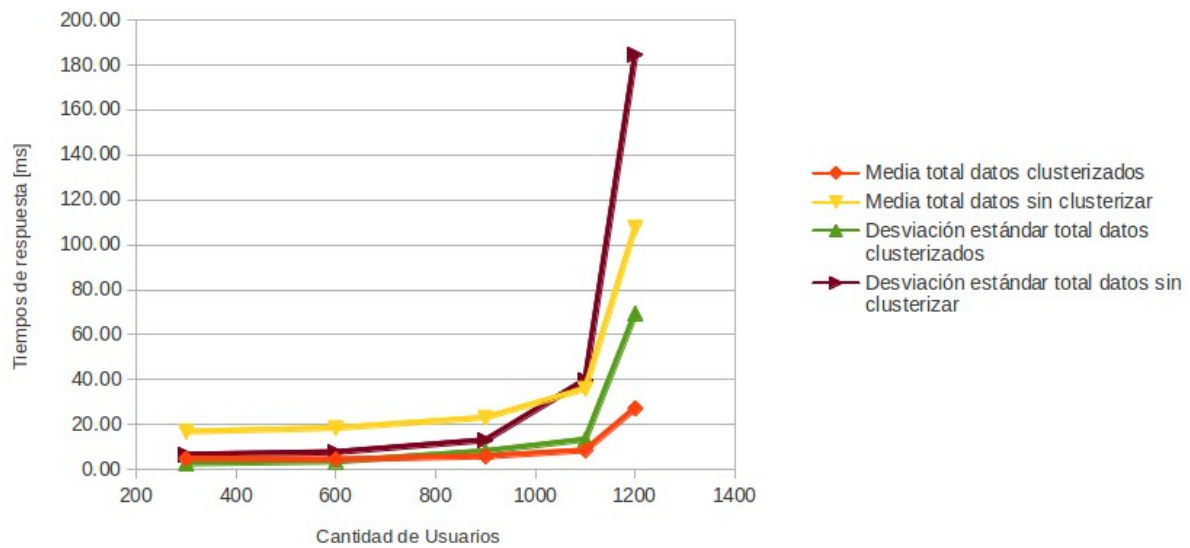
-explicar ejes

-comun de las pruebas dan asi

STT - JBoss - Cluster



STT - Terracotta - Cluster



- Diapositiva 15: Pruebas - Planillas (**Martín**)

- Guardar resoluciones + sincronizar texto
- Ver estadística y actualizarla
- Guardar un Trabajo Practico
- Guardar resoluciones + sincronizar texto
- Evaluar una resolución
- Guardar resoluciones + sincronizar texto
- Ver estadística y actualizarla
- Guardar un Trabajo Práctico

Descripción	Esta prueba le da 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
Capa de ejecución VC	Disponibilidad
Tipo de ejecución	X
Entrada	Automática
Salida	Escenario de pruebas y plan de pruebas "HttpProxy - Test Plan - Prueba completa"
Criterio de éxito	Tiempos medios, desviaciones estándar y rendimiento para cada tipo de petición
	Que los tiempos medios totales se mantengan dentro de los parámetros establecidos en la tabla de tiempos estándares

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
Capa de ejecución VC	Carga
Tipo de ejecución	X
Entrada	Automática
Salida	Escenario de pruebas y plan de pruebas "HttpProxy - Test Plan - Prueba completa Simple"
Criterio de éxito	Tiempos medios, desviaciones estándar y rendimiento para cada tipo de petición
	Que los tiempos medios totales se mantengan dentro de los parámetros establecidos en la tabla de tiempos estándares

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