# TESIS

tesis presentada ante la

**Universidad de la República, UdelaR**

para obtener el título de

MAGISTER EN INFORMÁTICA - PEDECIBA

por

Ing. Gustavo GUIMERANS

Instituto de Computación
Facultad de Ingeniería
UNIVERSIDAD DE LA REPÚBLICA

Título de Tesis :

## TESTING DE PERFORMANCE EN SISTEMAS CRÍTICOS: UNA NUEVA METODOLOGÍA Y APLICACIONES

a ser defendida en mayo de 2016 ante el tribunal

| | | |
|---|---|---|
| Franco | ROBLEDO AMOZA | Director Académico |
| Claudio | RISSO MONTALDO | Director de Tesis |
| Martín | VARELA RICCO | Revisor |
| Cristina | CORNES | Presidente |
| Andrés | ALMANSA | |

*Dedicado a David, María, Zulma y Serrana.*

# Agradecimientos

En primer lugar quiero agradecer a mi director académico Dr. Franco Robledo quien me animó a aventurarme en este desafío, guiándome sin permitir que me desconcentre ni desvíe y a mi director de tesis Dr. Claudio Risso por sus valiosos, adecuados y oportunos consejos.

También quiero agradecer a las instituciones que me permitieron llevar adelante este trabajo. Ellas son: CES (Centro de Ensayos de Software), FING (Facultad de Ingeniería), PEDECIBA (Programa de Desarrollo de las Ciencias Básicas), DL&A (De Larrobla & Asociados), Geocom, Financiera Confianza (Perú), IPContact. Este agradecimiento lo extiendo a Mariana Travieso, Mónica Wodzislawski (compañeros del CES), Álvaro Lamé, Héctor Cancela, Julieta López, Carlos Caetano, Gabriel Camargo, Jorge Abin y María Simon (directores del CES).

Con respecto a los casos de estudio, dónde radican importantes resultados hay algunos contribuyentes fundamentales. Remarco la participación de Edgardo Greising, Gustavo Vázquez (CES), Gustavo Steglich y Jorge Besil (DL&A), Roger Huapaya (Financiera Confianza), Jaime Bandeira, Federico Álvarez (Geocom), Amparo Mary, Gloria Magnifico, Ariel Sabiguero (ASSE), Jorge Merlino (IPContact) por permitirme reunirme participar en estos proyectos y reunir la información necesaria para comprender, aplicar y refinar la metodología.

Quiero extender mi agradecimiento al Dr. Martin Varela, Dra. Cristina Cornes, y Dr. Andrés Almansa por honorarme evaluando este trabajo.

No puedo dejar de mencionar a Laura González, Eduardo Grampín, Javier Baliosian, Gabriel López, Raquel Sosa y Diego Vallespir por aumentar mi interés y pasión por la investigación.

Por último, pero no menos importante a mi familia y amigos por apoyarme en este, y el resto de mis caminos. Por su amor, apoyo, paciencia y amistad. Este trabajo es dedicado a ellos.

# Resumen

El nuevo mundo es digital y crece a un ritmo sin precedentes. Se estima que hasta 2003 la humanidad había creado y almacenado digitalmente una cantidad de información equivalente a la que hoy se genera a diario. En la actualidad la mayoría de los procesos masivos, así como los datos y contenidos tanto públicos como personales, están informáticamente soportados. Por su creciente importancia y transversalidad a todos los sectores, los sistemas se han constituido en uno de los activos más críticos para las organizaciones. Buscando elevar la confiabilidad de esos sistemas, se recurre a diversas combinaciones de múltiples prácticas como ser: alta disponibilidad y performance de los componentes de los sistemas, procesos de desarrollo estandarizados y depurados en el tiempo, y el "testing de software", entre otros. El *testing* en particular busca una validación independiente sobre los requerimientos que un componente o sistema debe cumplir, y tiene múltiples variantes. En lo que hace al tipo de requerimiento destacamos: funcionales (si el resultado de las acciones es el esperado), performance (si soporta el nivel de carga o el volumen de datos necesario) y seguridad (capacidad para resistir fallas, o ataques intencionales). El objeto de esta tesis es introducir una metodología que sirva como marco de trabajo para realizar "pruebas de performance", y presentar además tres aplicaciones reales complementarias donde se constate su efectividad.

El "testing de performance" es un área de vanguardia, de alta complejidad, que requiere entre otras cosas el costoso desarrollo de una plataforma para interactuar con el sistema a probar. Es usual entonces que a la hora de priorizar pruebas, las organizaciones se inclinen hacia los aspectos funcionales, o incluso los de seguridad, en muchos de los cuales se puede avanzar sin enfrentarse a grandes dificultades tecnológicas. Esto es razonable para una pequeña empresa o para una aplicación con pocos usuarios o datos a manejar, pero es inaceptable en las grandes organizaciones, que son precisamente las que más dependen de la informática. Durante este trabajo no sólo veremos cómo aplicar la metodología a aplicaciones de distintos contextos tecnológicos, veremos además cómo los resultados de esas pruebas ayudan a optimizar el desempeño de los sistemas con mínimos ajustes en los componentes. Los casos son entonces evidencia de que incluso los sistemas soportados sobre componentes de hardware y software de tipo *world-class*, pueden no cumplir las condiciones mínimas para entrar en producción aún cuando hayan pasado por un proceso de validación funcional, y muestran también que la solución no necesariamente viene acompañada de inversiones en infraestructura.

La metodología aquí presentada fue co-desarrollada por el autor como miembro del Centro de Ensayos de Software (CES), a partir de las mejores prácticas existentes combinadas y ajustadas a la luz de la experiencia acumulada durante más diez de años en aplicaciones reales. Se elabora en actividades agrupadas en etapas, cuyo fin se resume en: identificar las transacciones representativas del uso esperado del sistema y los monitores para cuantificar su desempeño; la implementación de esas transacciones en un *framework* que permita automatizar la ejecución simultánea de combinaciones de múltiples instancias; la ejecución de varios ciclos de pruebas en los que se identifican los problemas a partir del análisis de los datos disponibles, se busca un diagnóstico y se repiten las pruebas explorando soluciones junto a los expertos del sistema. Durante los últimos diez años, distintas versiones de esta metodología han sido usadas en más de 20 organizaciones, algunas de las cuales atienden a más de 3000 usuarios, y ajustes de configuración mediante han permitido mejoras en los tiempos de respuesta del sistema de hasta 1000%. Entendemos que los resultados son alentadores y confiamos que se potenciarán por el creciente uso de sistemas distribuidos complejos, particularmente en la forma del denominado "cloud-computing".

**Palabras claves**: Testing de Performance, Metodología, Protocolos, Sistemas Distribuidos.

# Índice General

# Lista de símbolos

A continuación se presentan los símbolos y abreviaturas utilizadas en el documento.

## Abreviaturas

| Abreviatura | Término |
|---|---|
| AGI | Asterisk Gateway Interface |
| ANSI | American National Standards Institute |
| ATM | Automatic Transaction Machine |
| ASSE | Administración de los servicios de salud del estado |
| B2BUA | Back-To-back User Agent |
| CAPTCHA | Completely Automated Public Turing test to tell Computers and Humans Apart |
| CNG | Caja Nuestra Gente |
| CES | Centro de Ensayos de Software |
| CPU | Central Processing Unit |
| DL&A | De Larrobla & Asociados |
| FING | Facultad de Ingeniería |
| FTP | File Transfer Protocol |
| Gb/s | Gigabit por segundo |
| GNU | GNU's Not Unix |
| GPL | General Public License |
| HTTP(S) | Hypertext Transfer Protocol (Secure) |
| IBM | International Business Machines Corp. |
| IEEE | Institute of Electrical and Electronics Engineers |
| IIS | Internet Information Services |
| IMAP | Internet Message Access Protocol |
| IP | Internet Protocol |
| IT | Information Technology |
| IVR | Interactive Voice Response |
| JDBC | Java Database Connectivity |
| JDK | Java Development Kit |
| JMS | Java Message Service |
| JMX | Java Management Extensions |
| JVM | Java Virtual Machine |
| kb/s | Kilobit por segundo |
| LDAP | Lightweight Directory Access Protocol |
| Mb/s | Megabit por segundo |
| MOM | Java Message Oriented Middleware |
| PEDECIBA | Programa de Desarrollo de las Ciencias Básicas |
| POP | Post Office Protocol |
| POS | Point of  Sale |
| RAM | Random Access Memory |
| REST | Representational State Transfer |
| SIP | Session Initiation Protocol |
| SMTP | Simple Mail Transfer Protocol |
| SNMP | Simple Network Management Protocol |
| SOAP | Simple Object Access Protocol |
| SQL | Structured Query Language |
| SUT | System Under Test |
| SWEBOK | SoftWare Engineering Body Of Knowledge |

| | |
|---|---|
| TCP | Transmission Control Protocol |
| VTY | Virtual Teletype Terminal |
| W3C | World Wide Web Consortium |
| WAS | WebSphere Application Server |
| WMI | Windows Management Instrumentation |

# 1. Introducción

Determinar el correcto funcionamiento del software es intrínsecamente complejo, inabordable en forma general. Probablemente el resultado más representativo al respecto sea el *problema de parada* o *halting problem*, en el que se busca encontrar un programa que recibiendo a su vez como entrada: un programa cualquiera, su estado inicial y algún conjunto de parámetros de entrada, determine si el programa ingresado, a partir de ese estado y los datos que recibió, culminará en algún momento su ejecución o proseguirá indefinidamente. Basándose en resultados similares a los usados en 1891 por Alemán George Cantor para probar que los Números Irracionales no son Numerables, el Inglés Alan Turing demostró en 1936 que el *halting problem* no es computable, esto es, no puede implementarse en una computadora, o mediante un procedimiento. Al mismo tiempo emparentó la informática con algunas de las ramas más abstractas y trascendentales de las matemáticas, la lógica y la filosofía, ya que el resultado también está emparentado con el Teorema de Incompletitud de Gödel. La talla de estos temas y de sus respectivos autores nos da una primer perspectiva del enorme desafío oculto detrás del -en apariencia inocente- problema de probar software.

Lo anterior no implica que no puede validarse la correctitud funcional de ciertos grupos de programas específicos. Pero aún en esos casos, garantizar una salida esperada de un programa ante las combinaciones de conjuntos de estados iniciales y datos de entrada posibles, es una tarea también inabordable en general, en este caso por su costo en cómputo que a efectos prácticos es infinito. Cuando a los argumentos teóricos antes referidos se agrega el "factor de error humano" a lo largo del ciclo de vida del software: en especificaciones inadecuadas, errores de diseño o programación durante su construcción, durante la puesta en producción y en las tareas de mantenimiento posteriores, la certeza en la correctitud absoluta se vuelve una utopía. La manifestación más clara de esto se resume en el gran axioma del testing de Edsger Wybe Dijkstra: *"la prueba de un programa sólo puede mostrar la presencia de defectos, no su ausencia"*. Pero el testing de software no persigue la inútil y obstinada búsqueda de una quimera, sino orientarse hacia la utopía por el camino de lo humanamente realizable.

A lo que el testing apunta es a ponderar y elevar de ser posible la confianza en el software bajo evaluación. Su historia está repleta de anécdotas. Se considera al matemático inglés Charles Babbage como el padre de la computación moderna, ya que no solo redescubrió la máquina diferencial diseñada en 1786 por el alemán Johann Müller (una calculadora mecánica de polinomios), sino que entre 1833 y 1842 él mismo diseño la máquina analítica (construida 100 años más tarde), programable para hacer cualquier tipo de cálculo. Aún en el terreno de lo abstracto, se reconoce a Babbage, sus hijos, y a Ada Lovelace como los primeros programadores. Al haber encontrado errores en los cálculos de Babbage, a Lovelace se la reconoce además como el primer depurador de software (*debugger*).

El termino depuración (*debug*) se popularizó junto con el termino *bug* (bicho), utilizado para aludir a un error de software que desencadena un resultado indeseado. Aunque algunas anécdotas se remontan a Thomas Alva Edison, el hecho oficial en lo que a la informática refiere, data del 9 de septiembre de 1947 a raíz de un incidente con la computadora Harvard Mark II de la Armada de Estados Unidos, en la que un técnico solucionó la falla mediante la extracción de una polilla alojada entre los contactos de un relé.

En el resto de este capítulo repasaremos brevemente el objeto del testing, la evolución que ha tenido desde sus orígenes y las distintas formas de testing de software, con énfasis en el Testing de Performance y las razones de su creciente importancia.

## 1.1. Las pruebas o el testing de software

Aunque el testing está relacionado con la detección y eliminación de bugs, es bastante más que eso y ha evolucionado con el tiempo, diversificándose mientras acompañaba los cambios en la tecnología, la expansión de la informática y de sus aplicaciones. En 1979, Glendford Myers definió testing como el *"proceso de ejecutar el programa con la intención de encontrar defectos"*. Hoy en día se habla del testing con otros objetivos, relacionados a la búsqueda de calidad, más información para la toma de decisiones, mitigación de riesgos, entre otros. El mismo Myers plantea la separación entre el debug y el testing, y fija el espíritu del test en destruir: *"una prueba exitosa es aquella que encuentra un error"*, que puede tomarse como un corolario del ya citado axioma del testing de Edsger Wybe Dijkstra.

Dave Gelperin y William C. Hetzel en 1988 segmentan el testing de acuerdo a los cambios en sus fases y objetivos en su evolución. Su versión de la cronología es la siguiente: hasta 1956 todo era orientado a la depuración (revisar código al estilo Ada Lovelace), entre 1957–1978 se orienta a demostraciones, entre 1979–1982 pruebas orientadas a la destrucción, en el período 1983–1987 orientadas a la evaluación, mientras que en 1988–2000 las pruebas se orientan a la prevención. Hoy en día sigue orientado a la prevención y se agrega la evolución, la preparación a las constantes mejoras y nuevas funcionalidades que se agregan en general al software actual así como la portabilidad a distintas plataformas. Para seguir evolucionando y mejorando deberá incorporarse el testing en etapas tempranas del ciclo de desarrollo.

La verificación brinda información sobre la correcta elaboración del producto sea cual sea el paradigma o metodología, la validación permite discernir si el producto que se elabora es correcto. Lo anterior sugiere la diferencia de dos conceptos claves en lo que refiere a pruebas de software. Según IEEE 610.12-1990 de ANSI/IEEE (ANSI/IEEE) verificación es el proceso de evaluación de un sistema o componente para determinar si los productos de una determinada fase de desarrollo satisfacen las condiciones impuestas al inicio de la fase. Mientras que la validación es el proceso de evaluación de un sistema o componente durante o al final del proceso de desarrollo para determinar si se satisfacen los requerimientos especificados. Para ilustrar mejor lo que puede parecer un juego de palabras citaremos a otros autores. Sommerville habla de verificación cuando el fin es intentar comprobar que el sistema cumple con los requerimientos especificados (funcionales y no funcionales) ¿El software está de acuerdo con su especificación? Mientras que la validación intenta comprobar que el software hace lo que el usuario espera ¿El software cumple las expectativas del cliente? Boehm sintetiza los conceptos en la construcción. Para la verificación ¿Estamos construyendo el producto correctamente? Mientras que para validación ¿Estamos construyendo el producto correcto?

Volviendo al presente podemos referirnos al SWEBOK (SoftWare Engineering Body Of Knowledge) (Bourque & Fairley, 2014), un documento creado por la Software Engineering Coordinating Committee, promovido por la IEEE Computer Society, que se define como una guía al conocimiento presente en el área de la Ingeniería del Software y representa un amplio consenso respecto a los contenidos de la disciplina. Según el SWEBOK, una prueba es una actividad realizada para evaluar la calidad del producto y mejorarla, identificando defectos y problemas. La prueba de software es la verificación dinámica del comportamiento de un programa contra el comportamiento esperado, usando un conjunto finito de casos de prueba, seleccionados de manera adecuada dentro del dominio infinito de ejecución. El ser dinámica implica que para realizar las pruebas hay que ejecutar el programa para los datos de entrada. El comportamiento esperado permite decidir cuándo la salida observada de la ejecución del programa es aceptable o no, de otra forma el esfuerzo de la prueba es inútil. El comportamiento observado puede ser revisado contra las expectativas del usuario, contra una especificación documentada o contra el comportamiento anticipado por requerimientos implícitos o expectativas razonables.

La persona que interroga al producto de software para detectar fallas se la conoce como *tester*. El testing es una profesión desafiante, que demanda diversas capacidades, en torno al uso sistemas informáticos claro está, pero también en el relacionamiento y la comunicación con otras personas. El mismo *bug* como objeto, reviste ambigüedad, según quién y cuáles son las expectativas del interesado. Robert Sabourin plantea un concepto amplio de incidente (*bug*) que evita supuestos y mejora problemas de comunicación. Todo se vuelca como incidente y se discute entre los involucrados. Los incidentes son de diferentes niveles de gravedad e incluyen defectos, problemas o anomalías, inyectadas en un producto de software (por ejemplo en la etapa de definición de requerimientos, diseño, codificación) por omisión o por error y se presentan como una falla del software, pero también pueden ser un comportamiento emergente en un software que difiere de lo esperado. Con una visión que puede resultar extremista, Robert define un incidente como cualquier preocupación que alguien tiene sobre el proyecto en el que participa. Lo más importante y destacable es que los incidentes aportan información sobre la calidad del sistema.

El testing o las pruebas de software, son investigaciones técnicas y empíricas (Kaner, 2012) orientadas a proporcionar información sobre la calidad de un producto de software para un actor, usuario u organización, ya sea a efectos de adquirirlo o en el procesos de generarlo y mantenerlo. Uno de los desafíos del tester es cubrir la mayor parte del código o los componentes del sistema con la menor cantidad de pruebas. La cantidad de casos explorados, por sí solos, no resumen la efectividad de esas pruebas.

## 1.2. Tipos de testing de software

Las formas y métodos de testing son tan amplias como la misma informática. Ya se comentó la diferencia entre testing estático (análisis sintáctico y semántico del código) y dinámico (probar el programa ejecutándolo). También se segmenta el testing según el nivel de detalle interno disponible o deseable, yendo desde el white-box testing (donde se tiene pleno acceso a cualquier forma de estado, función o interface interna necesaria) hasta el black-box testing (donde sólo pueden inyectarse datos/eventos en la entrada de un componente y consultar su salida), pasando por distintos matices de gris.

Otra dimensión que se puede considerar es respecto al aspecto a evaluar en el software. En el lugar más visible encontramos al aspecto *funcional*, es decir: que el sistema cumple con sus especificaciones, con lo que se espera del mismo desde la perspectiva de los resultados obtenidos ante las entradas posibles. Dentro del *testing funcional* encontramos distintas estrategias para ejecutarlo, por ejemplo el *testing planificado* en el que inicialmente se diseñan casos de prueba y posteriormente se ejecutan los mismos, o el *testing exploratorio*, que implica que el aprendizaje del sistema y por tanto el diseño y la ejecución de las pruebas, se realizan de forma simultánea. Por otro lado se tienen las pruebas no-funcionales, donde se evalúan aspectos no relacionados a la especificación funcional cuando el sistema es usado por un único usuario en simultáneo. Dentro del testing no-funcional (o para-funcional) encontramos: performance (donde en lugar de los resultados obtenidos se busca evaluar los tiempos necesarios para conseguirlos), seguridad (donde se evalúa la probabilidad, impacto y posible mitigación de fallas, sean éstas accidentales o causadas por usuarios malintencionados), usabilidad, portabilidad, mantenibilidad, entre otros.

La continuidad de las pruebas dentro del ciclo de vida del software, y la madurez de este último, también generan categorías. La frecuencia con la que se requieren cambios y ajustes en los sistemas, y su complejidad, plantean desafíos a diversos actores, a los testers en particular cuando es crítico que esos cambios se den sin afectar el funcionamiento previo. Esto implica que el *tester* repita un conjunto representativo de pruebas (pruebas de regresión) para validar que la nueva *release* no introduce errores, al menos en lo que a ese conjunto de pruebas refiere. Esto está lejos de ser trivial. De hacerlo rigurosamente -algo necesario en grandes sistemas críticos-

termina siendo más costoso en esfuerzo que los mismos desarrollos. Una forma de mitigarlo es la *automatización de pruebas funcionales*, en las cuales tanto la ejecución de las pruebas como el procesamiento de los resultados obtenidos, se realiza en forma automática, por un sistema complementario al sistema bajo prueba. Es importante analizar el caso y su contexto con cuidado, porque la automatización de pruebas funcionales no siempre es la mejor solución. En definitiva la automatización también es software, particularmente scripts programados en algún lenguaje, que interactúan mediante interfaces programadas con el software bajo pruebas. Si no existe un núcleo estable con el que interactuar, los cambios en el sistema que afecten las interfaces, también forzarán cambios permanentes en la plataforma que efectúa las pruebas.

También se segmenta al *testing* por el tipo y tamaño de las unidades a testear. En este caso se habla de *testing unitario*, cuando las pruebas se hacen sobre los componentes más básicos, atómicos a efectos del sistema. Se pasa al *testing de integración* cuando se explora la combinación de módulos a través de sus interfaces, al *testing de interfaces* cuando se extienden las pruebas al pasaje de datos entre varias unidades o subsistemas, evaluando su interoperabilidad, hasta llegar finalmente al *testing del sistema*, que lo abarca punta-a-punta. En general esta secuencia de pruebas se limita a los aspectos funcionales. La siguiente etapa es la de las *pruebas de aceptación*. Éstas permiten validar que el sistema cumple con el funcionamiento esperado para ser aceptado y entrar en producción, desde todo punto de vista. Comprende tanto a los aspectos funcionales como a los no funcionales.

En términos de la caracterización detallada en esta sección, diremos que el objeto de esta tesis es introducir una metodología que sirva como marco de trabajo para realizar las *pruebas de performance* que forman parte de las *pruebas de aceptación.* Igual que en las pruebas de regresión funcionales, las pruebas de performance se sustentan usualmente en la automatización y tienen como premisa un mínimo nivel de estabilidad en las interfaces del sistema. Proponemos un enfoque dinámico (con el sistema funcionando), con interacciones sobre las interfaces del sistema para generar la carga, y al mismo tiempo de monitorización, registrando y cuantificando cómo esa carga afecta al sistema bajo prueba.

## 1.3. El testing de performance

El *testing* de *performance* busca determinar los tiempos de respuesta de un sistema ante diversas situaciones y condiciones de carga. El estudio de la performance de los sistemas es sumamente importante, indispensable antes de la aceptación; porque sin él podrían ponerse en producción sistemas lentos y carentes de utilidad práctica, aún siendo ellos funcionalmente correctos. Estas pruebas pueden además ayudar a minimizar perjuicios hacia otros sistemas distintos al testeado, evitando por ejemplo la interferencia en ambientes de multiprogramación o virtualización debidos a fugas de memoria. Las pruebas permiten identificar los componentes que consumen muchos recursos, localizar errores que solamente surgen notoriamente en situaciones de alta carga, obtener indicadores en requerimientos no funcionales, definir los límites que puede tolerar un sistema bajo determinada configuración, y explorar además formas de resolver esos problemas, que en algunos casos pueden ser tan simples, rápidos y baratos como cambios de configuración.

En otras ramas de la ingeniería como Civil o Aeronáutica, las pruebas de rendimiento son obligatorias en el proceso de construcción. La creciente importancia de los sistemas informáticos en actividades críticas, está volviendo al *testing* de *performance* más común cada día, pero su prominencia también es hija de los cambios de paradigmas en la tecnología. En los 80's las grandes corporaciones de IT (IBM y AT&T) proyectaban un mundo donde la informática se solucionaba con un puñado de mainframes, cuyos servicios serían ocasionalmente accedidos por los usuarios desde terminales tontas (VTYs), con interfaces ASCII que no requerían anchos de banda de más de 16kbps. La evolución de la telefonía digital fija se planeó sobre esas premisas

como medio para globalizar el acceso a la informática. Esta arquitectura cliente-servidor se acompañaba de un uso controlado de los recursos, ya que tanto los *mainframes* como la red telefónica y sus componentes, estaban inspirados en sistemas de tiempo real. En el peor de los casos se implementaban controles de acceso y de admisión, pero como norma, una vez conseguidos los recursos ellos estaban a disposición hasta terminar la tarea.

Por el contrario, tan solo tres décadas más tarde, la infraestructura de IT mundial está compuesta por miles de millones de potentes terminales (desde PCs de escritorio hasta *Smartphones*), conectadas por una red de extraordinaria velocidad (Internet). Como si ese antagonismo fuera poco, existen hoy día diversidad de sistemas operativos que soportan un número aún más diversos de aplicaciones, que tienden además a distribuirse geográficamente en una entidad denominada *la nube*. En pocas palabras, la nube es un conjunto de servicios globalmente accesibles mediante Internet, soportados en sistemas instalados en diversos *datacenters*, que tienden a funcionar sobre plataformas de virtualización.

Cuando en tan breve tiempo se ha pasado de una arquitectura de pocos servidores centrales en un entorno de recursos controlados, a una cantidad extraordinaria de componentes distribuidos, en la que el uso de los recursos (red, procesadores, memoria, discos) es compartido y su rendimiento por tanto estocástico, es natural que se genere un shock. Algunos apuestan a que el permanente aumento de potencia de cómputo y velocidades de comunicación, será la solución a los problemas de performance. Durante este trabajo presentaremos mediante aplicaciones reales, evidencia de que eso no es cierto como norma, ni eficiente como práctica.

### *1.3.1. Efecto de la arquitectura de un sistema en su performace*

Un requerimiento de performance es aquel que impone condiciones a un requisito funcional, como por ejemplo: la velocidad o el uso de memoria, con las cuales se debe ejecutar una función determinada. La descripción de los subsistemas y componentes de un software, así como el relacionamiento que mantienen entre ellos (ANSI/IEEE), es decir: *la arquitectura de un sistema*, es un insumo fundamental para poder estudiar su performance.

Independientemente de si fue intencionalmente definida o no, todo sistema de software tiene una arquitectura. Ella es el esqueleto del sistema e influye en los atributos de calidad a evaluar a través de elementos tales como: la interacción entre componentes y su distribución física, las estructuras de control, los protocolos de comunicación, la sincronización, el acceso a los datos. Desde la óptica del *testing* de performance, una de las diferencias más importante entre las distintas arquitecturas, es la distribución de los componentes de software que interactúan en un sistema.

Los distintos elementos de la arquitectura se pueden pensar como motores de procesamiento. Cada uno de estos motores podrá procesar distintas tareas en un determinado momento, utilizando ciertos recursos. Para estudiar la performance del sistema, será vital conocer el flujo de los datos por los distintos elementos de la arquitectura asociados a una solicitud y su respuesta y detectar dónde están los cuellos de botella. Generalmente agregar monitorización adecuada y aislar los componentes facilita su estudio y análisis. En consecuencia, la performance global del sistema será el resultado de la combinación de la performance de los distintos componentes de la arquitectura. Por tal motivo es importante conocer la arquitectura del sistema. Dependiendo de lo que se desee mejorar, deberán aislarse, estudiar o poner particular énfasis en unos u otros componentes. Mejorando la performance de cada componente se mejorará la performance global del sistema, porque los tiempos de respuesta que perciben los usuarios son la sumatoria de los insumidos en los distintos trayectos por los cuales fluye la información de una solicitud.

En una arquitectura monolítica el software se estructura en grupos funcionales muy acoplados. No hay distribución, ni a nivel físico (hardware) ni a nivel lógico (software); por lo que todo se ejecuta en una máquina, generalmente por un único usuario. Si una aplicación de escritorio tarda más de lo esperado en entregar resultados, el análisis podría pasar por monitorizar: CPU,

Memoria, uso de disco, etc. y ampliar el recurso más comprometido. Si esto no soluciona solo resta buscar optimizaciones en el software.

Una arquitectura Cliente-Servidor consiste básicamente en un programa (cliente informático) que realiza peticiones a otro programa (el servidor) para obtener cierta respuesta. Uno de los clientes más utilizados, es el navegador web. Muchos servidores son capaces de ofrecer sus servicios a través de un navegador web en lugar de requerir la instalación de un programa específico. El sistema cliente y el sistema servidor podrían estar en el mismo equipo aunque generalmente están conectados a través de una red de computadores. Si son varios los clientes que se conectan al mismo sistema servidor, toma la característica de sistema multiusuario. En esta arquitectura los componentes de software están distribuidos y es necesaria la existencia de una red para que los componentes se comuniquen entre sí. Esta comunicación es otro elemento a tener en cuenta cuando se perciben problemas de performance, llevando a analizar la velocidad con la que viajan las peticiones y las respuestas por la red. En sistemas con esta arquitectura, hay que tener en cuenta que no es suficiente probar con un solo usuario interactuando con el sistema. Tal vez sí se detecten problemas de performance cuando varios usuarios están accediendo, al mismo tiempo al sistema. Por ejemplo, se podrían sobrecargar las líneas de comunicación o los recursos del servidor. O quizás no se puedan entregar tantas respuestas como para cumplir con todas las demandas de los usuarios.

En un sistema con una arquitectura en capas cada una tiene responsabilidades definidas y encapsula un conjunto de funcionalidades relacionadas. La definición de capas puede variar dependiendo de las necesidades y elecciones de diseño de la aplicación, pero generalmente todas mantienen la división general de: “cliente – lógica – almacenamiento”, lo que da como resultado al menos tres capas. La capa de presentación o interfaz de usuario, a través de la cual los usuarios acceden al sistema. La capa lógica de negocio, en la cual se ejecutan los cálculos y se toman decisiones de ejecución según las reglas del negocio. La capa de almacenamiento de datos, por ejemplo los sistemas de bases de datos y sus correspondientes manejadores. Además de las arquitecturas de tres capas, pueden definirse capas intermedias con cometidos particulares, y se obtiene una arquitectura en N capas. Hoy en día las aplicaciones utilizadas por millones de usuarios (como algunas redes sociales) se sitúan en una arquitectura de este estilo. Los clientes acceden desde distintos dispositivos, por ejemplo móviles, terminales o computadores personales a una infraestructura que brinda el servicio que el cliente está demandando. El uso o no de determinados artefactos de la infraestructura, implicará la implementación de una arquitectura en una, dos, tres o N capas. Las arquitecturas cliente-servidor también se conocen como arquitecturas en dos capas.

En los últimos años surgió un nuevo paradigma en el área de arquitecturas orientadas a servicios el cual se conoce como *Cloud Computing*. Aquí las aplicaciones se ejecutan en ambientes remotos, generalmente virtualizados y accesibles a través de Internet; de los cuales no se conoce, a priori, su ubicación. En general los proveedores de estas nubes sugieren la despreocupación del uso de recursos y auto-escalamiento como una forma de dar respuesta a picos de uso y dificultad en el dimensionamiento de la infraestructura necesaria. Es importante destacar que si el sistema tiene fugas de memoria o procesamiento innecesario se estarán desperdiciando capacidades y por lo tanto se pagará más de lo necesario. Es éste uno de los motivos por el cual, en estas arquitecturas, las pruebas de performance también tienen un importante papel en las economías.

Se puede decir que en general la arquitectura va a determinar, entre otras cosas, dónde hay más procesamiento y dónde están los motores de la aplicación y por lo tanto los potenciales cuellos de botella que habrá que estudiar.

Los clientes informáticos (hardware y software) pueden ser clientes finos o gruesos. La principal tarea de los clientes finos es presentar la información que entrega el servidor y permitir al usuario generar nuevas solicitudes. En el cliente grueso existe procesamiento de reglas de negocio o

cálculos que hacen que el procesamiento sea mayor. El tipo de cliente con el que se accede al servidor determinará dónde están los motores de procesamiento.

Las arquitecturas utilizadas han variado con el paso de los años. Últimamente, es común el desarrollo de aplicaciones en más de dos capas, con un navegador como cliente fino que presenta al usuario lo que las otras capas procesaron. En este tipo de aplicaciones para probar la performance hay que definir y probar las capas que ejecutan del lado del servidor.

En un cliente grueso se tendrán motores de procesamiento a ambos lados la red que los interconecta y la performance está claramente dividida. Al pensar en la performance del sistema, hay que imaginar cómo optimizar determinado algoritmo, así como el procesamiento del lado del cliente y del lado del servidor. Integrantes del equipo de desarrollo pueden mejorar estos aspectos de performance en las distintas capas y componentes, probando unidades así como su integración, y midiendo los tiempos.

Es necesario identificar con cuidado el tipo de arquitectura para evaluar que tipo de simulación y herramientas serán requeridas para las pruebas de performance, ya es común que incluso aplicaciones tradicionales de escritorio como el MS Office, puedan crear y almacenar documentos online.

### *1.3.2. Tipos de pruebas de performace*

Para hacer testing de performance hay que tener en cuenta los elementos estructurales del sistema, ya que en éstos y sus interacciones radican los posibles problemas de desempeño (i. e. cuellos de botella).

Cuando se construye un sistema es importante pensar en la performance del sistema durante todo el desarrollo, y tomar decisiones para cumplir con las restricciones y los atributos de calidad requeridos o deseados. Se puede hacer testing de performance de cada algoritmo, componente o consulta a la base de datos por separado, así como también pruebas de performance sobre el sistema integrado, siguiendo el camino desde pruebas unitarias a pruebas de integración antes explicado. Aunque esto es lo recomendable, en general no se realiza y nuestro trabajo se centra en las pruebas de aceptación del sistema, una vez integrados todos los componentes.

Establecido ya el momento en que las pruebas serán realizadas, veremos ahora distintas variantes posibles, porque las particularidades y objetivos buscados en cada caso, sugieren distintos tipos de pruebas de rendimiento a realizar. Éstas se pueden clasificar en:

- **De Carga:** en las cuales se simula el nivel de carga media prevista para el uso del sistema.

  La carga de un sistema es el volumen de datos, o de actividad, que recibe en determinado lapso, está dada por la cantidad de clientes/sesiones/usuarios simultáneos y la actividad que cada uno genera. Se estudia el tiempo de respuesta (rendimiento). Se busca mitigar los riesgos de que no se soporte la carga esperada en cuanto a usuarios y uso del software. Es el nivel mínimo a superar para la aceptación.

- **De Stress:** en las cuales se lleva al sistema a límites de carga que exceden los previstos en su uso. Mitigan los riesgos de un crecimiento no esperado o contemplado que en algunos casos se da por la fusión u adquisición de organizaciones, así como la acumulación de trabajo o situaciones extremas. Caracterizan la robustez del sistema.

- **De Escalabilidad:** que es un caso particular de una prueba de carga, en la cual se estudia el rendimiento cuando se disminuyen o aumentan los recursos físicos del sistema.

  Son un insumo esencial para planear el crecimiento de infraestructura necesario para acompañar al ciclo de vida de al aplicación.

- **De Volumen:** que es una prueba de carga en la cual se estudia el rendimiento cuando se incrementa el volumen de datos en los repositorios del sistema. Son comunes los casos donde

las pruebas de carga con los repositorios de datos vacíos dan resultados satisfactorios, luego, al tiempo el sistema se degrada a medida que acumula datos.

Estas pruebas permiten dimensionar el hardware asociado al almacenamiento, definir la mejor configuración en cada escenario en lo que respecta principalmente a parámetros de los motores de las bases de datos.

Si bien en la industria se manejan estas categorías de pruebas de rendimiento, lo más importante a tener en cuenta a la hora de pensar en una prueba de rendimiento no es en qué categoría cae sino qué preguntas debe responder, qué información se desea obtener y qué riesgos se quiere mitigar. En general el proceso de pruebas va acompañado de ingeniería de rendimiento en la cual, de existir, se estudian los problemas y se busca solucionarlos.

Elegida la variante de prueba de performance a realizar y suponiendo que se está en el estado previo a la puesta en producción de un nuevo sistema, la aproximación más directa para generar carga y evaluar la respuesta pasa por la ejecución en forma manual. Se podría solicitar, por ejemplo, a tantos usuarios (empleados) como se deseen simular que concurran un día en el que no tengan que trabajar (para que no se tenga que detener la operativa) y ejecutar las funcionalidades a probar. Para esta simulación existen varios desafíos. El primero es que concurran todos los usuarios necesarios para la prueba, pueden ser cientos o miles. Además cada operario necesita un equipo para operar. El segundo es coordinarlos para simular el comportamiento buscado para la prueba de performance. Será necesario elaborar guiones para que los usuarios sigan durante la prueba. Finalmente será necesario analizar los resultados obtenidos, accediendo a las bases de datos, monitores del sistema, monitores de red, entre otros, pero también saber la opinión de los usuarios acerca de su experiencia con el sistema (la cual será subjetiva). Por ejemplo, si se desea hacer una prueba de carga, se podría necesitar que 500 usuarios ejecutaran una funcionalidad simultáneamente, en el mismo instante. En estos casos, se necesitaría un coordinador que emita una señal para que todos los usuarios hagan clic en algún vínculo o botón al mismo instante. Si se logra avanzar para lograr una buena prueba, será producto de un gran esfuerzo colectivo, existe la posibilidad de que se produzca un incidente invalidante, que no había sido detectado en las pruebas ejecutadas anteriormente (de integración, funcionales). En ese caso, habrá que cancelar y duplicar todo el esfuerzo. Pero no sólo si se detectara ese evento invalidante habría que repetir, también si se desea realizar una nueva prueba luego de un cambio o si se desea agregar más usuarios.

Otro problema radica en la asignación uno a uno de los usuarios con equipos, que no siempre son computadores personales. Cuando algunos de los terminales son, por ejemplo, cajeros automáticos sobre una red de pagos sería inviable tomarlos todos para pruebas.

Las pruebas de *performance* basadas en simulaciones con usuarios son excepcionales y complementarias a las pruebas de *performance* automatizadas ya que permiten simular más casos pues su ejecución no implica automatización sino definición y asignación. Cuando se llevan adelante, son puntuales y generalmente motivadas por objetivos de capacitación hacia los funcionarios. En 10 años de experiencia fueron tres las veces que participamos de simulaciones con usuarios reales. Otro caso similar es cuando los problemas se dan con el sistema en producción o en los primeros pilotos, generalmente el funcionamiento del mismo ya es crítico para la operativa de la empresa y la opinión que los usuarios se formen es crucial para la adopción de la nueva herramienta por parte de los mismos. En estas circunstancias, los tiempos disponibles de análisis suelen ser mínimos y el interés de las partes ante un incidente pasa por retornar el nivel de servicio requerido en el menor tiempo posible. Sin embargo, la estabilización del sistema no se dará hasta que estos problemas sean atacados y solucionados. Por esta razón es que se vuelve sumamente interesante contar con un monitoreo objetivo y continuo, apoyado en herramientas de monitorización que nos permita realizar esta tarea de manera desatendida para encontrar las causas de los problemas de performance, determinar los posibles cuellos de botella que se pueden dar y determinar las causas de caídas o “pegadas” en el sistema.

En los trabajos sobre sistemas en producción se puede seguir una metodología similar a la utilizada durante la ejecución de las pruebas de performance. Esta metodología se basa en la recolección de datos y el análisis de los mismos con la finalidad de identificar puntos de mejora y la aplicación de los mismos de manera ordenada y efectiva. Mediante la monitorización a nivel de infraestructura (red, sistema operativo, manejador de base de datos, etc.) y de las funcionalidades y tiempos de respuesta de las aplicaciones, se busca obtener la monitorización a todo nivel para así poder contar con una visión global y objetiva de la situación.

En los casos antes mencionados generalmente la etapa de automatización no se realiza. La metodología presentada en esta tesis asume que hay automatización.

Las herramientas para generación automática de carga permiten ejecutar una y otra vez las mismas o diferentes pruebas, incluso con diferentes objetivos. Evitan la necesidad de contar con operarios, puede alcanzar con un *tester*. Se pueden simular diferentes escenarios de carga, variando la cantidad de usuarios y las funcionalidades que ejecutan. Por ejemplo se puede ejecutar una prueba con diez usuarios y en caso de ser válida elevar la cantidad de usuarios a 100, luego a 200, así hasta alcanzar varios miles de usuarios concurrentes, de ser necesario. Y en cada instancia, analizar cómo se comporta el sistema. Otra ventaja es que las pruebas son reproducibles automáticamente, ya que bajo las mismas condiciones se ejecutan exactamente los mismos pasos con los mismos puntos de validación que se programan, incluso cuando se está generando una alta carga, lo cual es algo difícil de lograr en una prueba manual con muchos usuarios. Además, estas herramientas permiten conocer en un momento dado, con precisión, cuántos usuarios están ejecutando y qué funcionalidad. Un beneficio a destacar, es que las pruebas se pueden ejecutar de forma desatendida, por ejemplo de noche, y analizar los resultados a la mañana siguiente. Pero también existen riesgos y limitaciones. Para entregar datos precisos, es importante garantizar que los equipos y la herramienta utilizados para generar la carga, no presenten problemas de performance, por lo cual se monitoriza su comportamiento.

La generación de las pruebas automatizadas también tiene costos asociados como la curva de aprendizaje de la herramienta, la gestión de los cambios cuando se modifica la aplicación, entre otros. Por lo tanto, es preciso seleccionar y acotar el conjunto de funcionalidades a automatizar. Para seleccionar estas funcionalidades (también llamadas transacciones en el ámbito de las pruebas de performance), se identifican cuáles son las más críticas, las más usadas y en particular cuáles pueden tener problemas de performance en el contexto de la prueba. Esto agrega una nueva diferencia con la realidad de uso, es decir, la realidad deberá ser emulada con un conjunto acotado de funcionalidades y no con el total.

Las herramientas facilitan el modelado y la ejecución de las pruebas de carga. El mayor inconveniente que presenta este tipo de pruebas es que cada aplicación debe ser estudiada concretamente, y se debe elaborar un modelo de carga que sea consistente y representativo del uso real de la aplicación, para que de esa manera los resultados obtenidos a posteriori provean información útil. Ésta no es una tarea fácil y a menudo es la que lleva mayor tiempo del proceso o que puede invalidar una prueba o mantenerla alejada de la realidad que pretende representar.

Una vez han sido generados dichos modelos de carga, se procede a implementar scripts de usuarios virtuales que son ejecutados sobre la aplicación de forma dinámica, donde se monitorizan métricas definidas y se colectan datos para luego ser analizados.

Existen metodologías y herramientas específicas para dar soporte a este tipo de pruebas que difieren respecto a otros tipos de pruebas, como las funcionales. A modo de ejemplo en una prueba de performance es sumamente crítico el conocimiento de la infraestructura en que se ejecutan las pruebas mientras que en el testing funcional puede no serlo. La importancia radica en el estrecho vínculo que existe entre el hardware y el software bajo pruebas. Al ser finitos los recursos de una infraestructura donde el sistema es desplegado; encontrar los límites en una configuración será distinto a encontrarlos en otra que contenga diferencias, por más que éstas

sean a priori sutiles. Lo antes mencionado sugiere la importancia de analizar el contexto donde ejecutará el software y por lo tanto el ambiente para ejecutar las pruebas de performance.

Existen varias decisiones que el líder de testing debe tomar previo y durante el desarrollo de un proyecto de performance. La clave de esas decisiones se basa en identificar cuál es el requerimiento concreto y plantear la mejor prueba a realizar. A modo de ejemplo, es muy importante saber qué tipo de pruebas se pueden realizar en la etapa que se encuentra el sistema y cuál es la más adecuada, así como la información que brinda cada tipo de prueba, el tiempo que demanda su realización y las variables asociadas. Un proyecto de performance puede tener muchas dependencias que pueden dificultar el avance, por lo tanto es importante tener un plan para esas situaciones. También puede tener muchos supuestos que podrían tener alternativas. Por ejemplo, usualmente sucede que entre la detección del problema y la liberación de la versión del sistema que supuestamente tiene implementada la mejora, podrían pasar semanas. También es necesario seleccionar o implementar una herramienta y conocer las responsabilidades de los actores.

## 1.4. Motivación y propuesta de valor

El testing de performance busca determinar los tiempos de respuesta de un sistema ante diversas configuraciones y situaciones de carga. Se busca obtener información precisa que sirva para la toma de decisiones, cuantificar las capacidades de las infraestructuras y validar los requerimientos de rendimiento y escalabilidad de las plataformas y del sistema a probar. Algunas de las preguntas para las que se busca respuesta son:

1. ¿Qué cantidad de usuarios simultáneos soporta el producto con tiempos razonables de respuesta sobre la infraestructura y plataformas propuestas?
2. ¿Es suficiente el hardware para soportar el nivel de transacciones propuesta?
3. ¿Qué expectativa de crecimiento puede soportar?
4. ¿La performance lograda bajo esta plataforma es la deseada?

Estos resultados son fundamentales para liberar el sistema al ambiente producción, mejorar su rendimiento en los distintos entornos donde se aplique y planificar su crecimiento.

*Las principales contribuciones de esta tesis son: i) una metodología para realizar pruebas de rendimiento de distinto tipo sobre sistemas críticos, de probada eficacia en aplicaciones reales; ii) la presentación de una serie de casos de estudio donde se aplicó la misma, que constituyen evidencia de los resultados alcanzables con la metodología y al mismo tiempo ejemplos de cómo enmarcarla en problemas reales. Es importante destacar que el trabajo se limita a las pruebas de performance. Otras pruebas, como las funcionales o de seguridad están fuera del alcance de esta tesis.*

La metodología aquí presentada fue co-desarrollada por el autor como miembro del Centro de Ensayos de Software (CES) a lo largo de diez años. Se elaboró a partir de las prácticas existentes, que fueron combinadas y ajustadas en base a aplicaciones reales. La misma se resume en: identificar las transacciones representativas del uso esperado del sistema y los monitores para cuantificar su desempeño; la implementación de esas transacciones en un *framework* que permita automatizar la ejecución simultánea de combinaciones de múltiples instancias; la ejecución de varios ciclos de pruebas en los que se identifican los problemas a partir del análisis de los datos disponibles, se busca un diagnóstico y se repiten las pruebas explorando soluciones junto a los expertos del sistema.

Los casos de aplicación presentados en este documento corresponden a proyectos liderados por el autor, y han sido seleccionado a efectos de cubrir una muestra representativa de tecnologías, tipos de prueba, y características complementarias del uso de la metodología.

La lista de casos es la siguiente:

1. La fusión de dos bancos en Lima (Perú), que implicó integrar productos financieros existentes en una de las instituciones sobre el *core* bancario de la otra, que debía además absorber los nuevos clientes y sucursales.
2. El segundo caso de estudio describe las actividades realizadas en el marco de un proyecto de test de performance de un sistema de central telefónica y *contact center*, con el fin de incrementar su potencia a efectos de ser un producto a un segmento de clientes de mayor tamaño.
3. El tercer caso de estudio describe las actividades realizadas en el marco del proyecto de test de performance de un sistema web de gestión de consultas médicas, para una institución que deseaba mantener tiempos de respuesta aceptables en un escenario de expansión del sistema a todos el país.

Cada caso será presentado como un capitulo en sí mismo.

## 1.5. Trabajos relacionados

Fueron detectadas en la literatura metodologías o aplicaciones similares a las aquí presentadas. Algunas incluso han servido como inspiración para este trabajo. A continuación se enumeran y detallan sus características y diferencias.

En el contexto histórico y en línea con nuestro trabajo comenzamos por citar artículos de Elaine Weyuker. En (Avritzer & Weyuker, Generating Test Suites for Software Load Testing, 1993) los autores presentan un esquema para la generación automática de casos de pruebas de carga. En (Avritzer & Weyuker, Deriving Workloads for Performance Testing, 1996) abordan la problemática de comparación de cargas, particularmente para el reemplazo de arquitecturas. Un artículo de referencia es (Vokolos & Weyuker, 1998); en el mismo los autores discuten enfoques de pruebas de rendimiento de software, así como un *framework* y un caso de estudio que describe una experiencia concreta en pruebas de rendimiento. El trabajo fue revolucionario para su fecha, pues entonces era poco el material publicado sobre pruebas de performance. Su *framework* incluye: i) el diseño de un algoritmo para generar casos de prueba específicos para probar aspectos de performance; ii) la definición de métricas para cuantificar la exhaustividad y comparar la efectividad de los algoritmos de selección de casos de prueba para un determinado programa; iii) la definición de relaciones para comparar la efectividad relativa a diferentes estrategias de pruebas de performance en general y iv) la comparación de diferente hardware o arquitecturas para un determinado programa. Por aquel entonces tampoco había un protocolo estándar y de hecho el software del caso de estudio incluía un Gateway para soportar una mezcla de protocolos (TCP/IP, SNA/3270 y TN3250), y se implementaban programas *ad-hoc* (caso a caso) para la generación de carga. Si bien los autores no describen claramente la metodología que siguen, presentan las dificultades para modelar la realidad y el detalle de las hipótesis formales que asumen para las pruebas. Posteriormente Weyuker ha participado en otros trabajos relacionados donde se destaca la importancia del testing de performance, como son los realizados con Avritzer, tal como (Avritzer & Weyuker, The role of modeling in the performance testing of e-commerce applications, 2005) para aplicaciones de comercio electrónico y (Avritzer, Bondi, & Weyuker, Ensuring system performance for cluster and single server systems, 2007), donde se propone un enfoque para eliminar la degradación de la performance usando una métrica basada en la experiencia de usuario, que al compararla frecuentemente con el objetivo de performance pueden asegurar la estabilidad a un bajo costo mientras que en (Avritzer, Kondek, Danielle, & Weyuker, 2001) el enfoque se basa en carga de trabajo.

Por encima de aspectos metodológicos particulares, los trabajos de Weyuker antes citados comparten con el nuestro el objetivo general: elevar la confiabilidad de un sistema mediante un

enfoque preventivo. En general la prueba exhaustiva de un sistema no es viable ni conveniente, así que las pruebas de performance buscan bajar la probabilidad de aparición de eventos negativos durante el funcionamiento de un sistema. Un enfoque complementario pasa por suponer que tales eventos son intrínsecos al software, y pensar entonces en cómo mitigar sus efectos en lugar de buscar prevenirlos. En (Avritzer, Bondi, Grottke, & Weyuker, 2005) se presentan algoritmos para el "rejuvenecimiento del software", un enfoque para ayudar a prevenir la degradación y otras fallas asociados que se presentan con el tiempo. Esta técnica fue identificada como una solución rentable durante la investigación en el AT&T Bell Laboratories en la década de 1990 (Cotroneo, Natella, Pietrantuono, & Russo, 2014). El método más común es el famoso reinicio del sistema, que hoy en día se realiza de forma programada en máquinas virtuales que ejecutan en la nube. Otro ejemplo es la implementación del servidor web Apache donde se matan y recrean los procesos después de un cierto número de solicitudes (Bruneo, Distefano, & Longo, 2013). En noviembre de 2015 se organizó la 26th IEEE International Symposium on Software Reliability Engineering donde se organizó el séptimo International Workshop on Software Aging and Rejuvenation, lo que muestra que esta aproximación al performance testing sigue teniendo relevancia. Aunque útil en ciertos contextos y complementaria a la metodología aquí presentada, el rejuvenecimiento del software no da solución a todos los problemas (a los de volumen por ejemplo) y tampoco es siempre aplicable en sistemas con componentes críticos.

La metodología presentada en esta tesis se inspira en los lineamientos recogidos en una serie de artículos de Scott Barber, titulados "User Experience, Not Metrics" (Barber, 2006). Los lineamientos planteados por Barber se basan en la simulación del comportamiento esperado, y persiguen la detección y corrección de incidentes, con el fin de prevenirlos antes de liberar el sistema a producción. Este material es la referencia por excelencia para diseñar la simulación de la realidad por medio de la automatización de usuarios virtuales. Los primeros capítulos (2, 3 y 4) tratan sobre el modelado realista de usuarios a través de la codificación de scripts. Cada script simulará un usuario, por lo que ejecutándolo varias veces de forma concurrente se simula una cantidad determinada de usuarios/carga. Para lograr un comportamiento representativo de un usuario real se introducen retardos artificiales para simular el procesamiento en el cliente o agregan pausas (*sleeps*) para simular el lapso en que los usuarios humanos no están interactuando directamente con el SUT, antes de realizar una acción sobre la aplicación. Estos “tiempo de pienso” (*think times*), inherentes a los usuarios humanos por ser seres pensantes, simulan los tiempos dedicados por el usuario a tareas tales como leer el contenido de la pantalla, seleccionar un objeto de una lista, llenar un formulario, ingresar información, hablar con el cliente que se está atendiendo detrás de un mostrador, entre otros. Luego de tener codificados todos los scripts que se utilizarán, es necesario que se ejecuten en conjunto de alguna forma, simulando una comunidad de usuarios real utilizando el SUT. Finalmente, se monitorea el tráfico y la interacción de estos usuarios virtuales con la aplicación y se obtienen conclusiones sobre la performance esperada del sistema.

Una parte crítica del *testing* de *performance* es obtener medidas útiles y fidedignas, ya que de no ser representativas pueden desalinear los resultados y desorientar respecto adónde poner esfuerzo para mejorar el sistema.

Los *timers* son los elementos presentes en los *scripts* que permiten obtener los tiempos de respuesta de los diferentes intervalos a medir durante las pruebas. Se incluyen *timers* para cada una de las transacciones completas así como también para cada uno de los pasos individuales que las conforman. Para estudiar la *performance* es útil, por ejemplo, revisar su evolución en distintos escenarios, a lo largo del tiempo y con distintos niveles de concurrencia. Para esto, los *timers*/monitores deben estar en el lugar correcto y sus mediciones deben además ser correctamente interpretadas. En la mayoría de las gráficas que se presentan, se visualizan los datos obtenidos por estos *timers*. Los siguientes capítulos de Barber (5, 6 y 7) se detienen justamente en cómo medir correctamente los tiempos de respuesta. Los lineamientos integran al

análisis la naturaleza estocástica del proceso que determina la performance de un sistema punta-a-punta. Así Barber se apoya en criterios estadísticos para validar las mediciones, identificando valores anormales (*outliers*), fijando criterios para descartarlos en un principio, pero también para integrarlos en las mediciones si aparecen frecuentemente. Estos capítulos fijan recomendaciones respecto a cómo consolidar y presentar los resultados de los *tests*, a efectos de que tengan validez estadística y recojan los principales datos de la simulación. Citando a Barber: "*Dos ejecuciones de una prueba serán idénticas si la prueba en sí (mismos parámetros para misma cantidad de usuarios) y el ambiente son idénticos entre ambas pruebas. Se pueden desarrollar scripts con cierto grado de aleatoriedad manteniendo la equivalencia de las pruebas. La clave para determinar si las pruebas son idénticas es simplemente si los resultados de las pruebas son los mismos. Calcular matemáticamente la significación estadística, la equivalencia, o correlación entre las muestras está fuera del alcance. Un enfoque matemático que implica Chi-cuadrado, t-test, o métodos de p-valor, se puede encontrar en* (Statsoft-Dell Software)*".*
El análisis estadístico de los resultados, es en sí mismo un problema complejo, de actualidad en la academia. Los ambientes reales tienden a ser virtualizados en *datacenters*, de donde la aleatoriedad proviene no sólo de la interacción con los clientes sino de los recursos disponibles. Aunque más completa, para ser representativa (i.e. margen de error pequeño) una prueba aleatorizada requeriría un tiempo de ejecución tan extenso que se volvería impráctica.

El contexto en el que se dan este tipo de proyectos es usualmente, y en este caso particular, al finalizar la etapa de desarrollo, luego de realizar el testing funcional y teniendo un producto funcionalmente estable. Generalmente los cronogramas son exigentes y asumen que no habrán problemas que dificulten el avance, las pruebas funcionales detectan errores que de llevarlos a producción puede tener consecuencias severas, la corrección de estos incidentes no es inmediata y puede inyectar errores en otras funcionalidades (regresión del software) lo cual hace necesario ejecutar nuevas pruebas.

En la práctica es inviable automatizar todos los flujos del SUT y es por ello que en los hechos una prueba viable en tiempo, generalmente, no escala a más de 10 transacciones aunque con más flujos sería más rica. Es común que el acuerdo de las transacciones a incluir sufra cambios respecto a un planteo ideal y disminuya, de hecho hay que ser selectivo con las transacciones seleccionadas, hay casos en los que de diez se ha disminuido a dos transacciones. Si bien esto aleja la simulación de la carga real ya que el escenario completo incluiría otras transacciones es mejor que no realizarlo y permite detectar muchos problemas. De otra manera el tiempo no alcanza para detectar si el software soportará, al menos, algunos flujos.

Una prueba de carga mal realizada puede llevar a subestimar o sobreestimar la capacidad de una aplicación. Además hay pruebas especialmente apropiadas para evaluar ciertos aspectos, en distintos momentos del proceso de desarrollo. Los siguientes capítulos de la serie de Barber (8, 9 y 10) se detienen en recomendaciones sobre qué pruebas realizar para tener una simulación representativa y en cómo reportar los incidentes para que maximizar la eficacia de las correcciones a realizar. Algunos de los tests sugeridos por Barber y su propósito específico constan en la siguiente lista a modo de ejemplo:

i) Test Base/Baselines: son tests mono-usuario empleados para validar la funcionalidad de un script y tener una noción de la performance general del sistema. También permiten establecer los tiempos de respuesta asociados al "mejor caso", que es cuando hay un único usuario realizando una única transacción.
ii) Test Benchmark: emplea un escenario involucrando una pequeña comunidad de usuarios. Aseguran que el entorno de *testing* se comporta como se esperaba bajo cargas livianas, y también valida que los scripts se han implementado correctamente. También sirven como comparación para futuros *tests*.
iii) Tests basados en componentes: generan carga sobre un componente o capa del sistema y comúnmente son utilizados para verificar los resultados de realizar ciertos ajustes (*tunings*).

Hay tres tipos principales de medidas de componentes útiles: tasa de transacciones, uso de memoria y uso de CPU.

iv) Tests de tiempo de respuesta: simulan usuarios reales interactuando con el sistema (incluyendo *thinking times*) para estimar el tiempo esperado de respuesta. Se realizan ciclos de distinta carga. En cada ciclo se eleva gradualmente la carga (*ramp-up*), se sostiene durante un período estacionario, y luego se decrementa hasta cero. Se comienza con cargas bajas, e incluso sobre escenarios parciales, y se incrementa gradualmente buscando aislar patrones en la performance.

v) Tests de Escalabilidad: Son utilizados para determinar cuántos usuarios reales pueden acceder al sistema antes de que los tiempos de respuesta del sistema se tornen inaceptables. Generalmente, son exactamente las mismas pruebas utilizados para estudiar los tiempos de respuesta, ejecutados con pequeños incrementos del número de usuarios virtuales. Otro tipo de *tests* de escalabilidad estudia el comportamiento del sistema ante cambios en la infraestructura base y en los recursos disponibles, es decir, hardware y software. No se usan para determinar si un sistema fallará, sino para saber dónde fallará primero, cuán serio será el fallo y por qué.

vi) Los *Spike Tests* (test de picos): son una variante de la anterior, con cargas reales, pero con tiempos de *ramp-up* y *ramp-down* extremadamente rápidos. Comúnmente hacen *ramp-up* hasta 100% o 150% del pico de carga esperado en 10% del tiempo normal de *ramp-up*.

vii) Pruebas de Estrés: cargas reales pero bajo condiciones extremas (nivel de carga máximo durante mucho tiempo). Son ejecutados generalmente luego de haber hecho varios ajustes (*tunings*). Permiten detectar defectos sutiles (como fugas de memoria). Si todos los componentes siguen funcionando bien, con tiempos de respuesta razonables, y se recuperan apropiadamente luego de la ejecución del test, pasan el test de estrés.

viii) Test de Hammer: cargas reales eliminando los "user thinking times", e incrementado la simultaneidad hasta que ocurre una falla. Son designados para encontrar los "puntos de quiebre" (*breakpoints*) de un sistema de manera de poder obtener estrategias de mitigación de riesgo. Complementan los Tests de Escalabilidad, explorando situaciones más extremas e improbables.

En grandes líneas Barber ubica las métricas de performance desde la perspectiva de los tiempos percibidos por los usuarios, pero propone recomendaciones sobre cómo presentar los reportes de resultados para que sean útiles desde la perspectiva de gerentes y/o accionistas (*stakeholders*), en el entendido que más que técnica ésta es información de negocio, útil para decidir si se libera un sistema a su explotación comercial, o si se da el visto bueno para adquirirlo desde un tercero. Los capítulos 8, 9 y 10 de Barber fijan una serie de recomendaciones respecto cómo combinar: texto, tablas y gráficos a efectos de resaltar las principales conclusiones de las pruebas para este segmento específico de lectores. En los capítulos posteriores de este documento mostramos ejemplos de cómo seguir las recomendaciones de documentación de Barber.

Los últimos capítulos de la serie (11, 12 y 13) tratan temas técnicos avanzados, como IDs de Sesiones Seguras, etc. Sin entrar en detalles diremos que son cada vez más frecuentes las comunicaciones encriptadas entre los componentes de un sistema distribuido, y/o aquellas que requieren algún nivel de autenticación. En estos capítulos Barber incluye recomendaciones sobre cómo interactuar con sistemas de estas características, poniendo énfasis en la tecnología WEB: HTTP, su estándar de encriptación (HTTPS), sus mecanismos de autenticación automática (cookies) y el manejo de sesiones.

En algunas pruebas es posible desactivar la autenticación y la encriptación, pero en otras o bien no es posible o no es deseable, lo que puede incrementar notoriamente la complejidad, especialmente en los desarrollos de scripts o la captura de datos. Sin embargo es creciente el número de sistemas distribuidos (especialmente los comerciales), que se comunican haciendo uso de tecnología WEB, por lo que entender su funcionamiento en detalle resuelve la mayoría de

los casos. El libro (Meier, Farre, Bansode, & Barber, 2007) es la referencia más completa sobre el tema. Abarca temas introductorios y sus fundamentos metodológicos, tanto del proceso de testing como de su aplicación en distintos contextos.

Para finalizar esta sección repasamos las líneas de trabajo de Che y Maag. El primer artículo que merece ser tenido en cuenta es (Che & Maag, 2013), donde los autores presentan un enfoque basado en la lógica para probar el rendimiento de un protocolo a través de trazas de ejecución reales y especificando propiedades formalmente. Con el fin de evaluar y valorar su metodología, desarrollaron un prototipo y presentan experimentos. Utilizan la herramienta SIPp y confirman la arquitectura maestro esclavo para la generación de carga. La evaluación de las propiedades retornan éxito, falla o resultado inconcluso. El resto de las publicaciones de Che, en general con Maag, son actualizaciones, complementos y mejoras de éste. El enfoque se basa en testing pasivo es decir en la observación de eventos de las entradas y salidas de una Implementación Bajo Pruebas (IBT o IUT del inglés) en tiempo real. Se diferencia del testing activo porque no modifica el comportamiento natural del IBT y por lo tanto no tiene sus ventajas (ej. cobertura de los casos) se puede complementar con monitoreo. El trabajo inicial es (Lalanne, Che, & Maag, 2011) y marca las características de los trabajos (similares) sobre pruebas formales, en la mayoría haciendo incapié en la performance, por ejemplo (Che & Maag, Formally testing the protocol performances, 2014), (Che & Maag, Passive Performance Testing of Network Protocols, 2014), (Che, Maag, López, & Cavalli, 2014), (López, Che, & Maag, 2014) las principales contribuciones son el enfoque formal propuesto para expresar los requerimientos de conformidad (se habla de Implementación Bajo Pruebas) y performance y el diseño de un framework para realizer su testing en tiempo real. En el caso de (Che & Maag, 2013), basado en (A Passive Testing Approach for Protocols in Internet of Things, 2012), el foco es el internet de las cosas y el framework prueba servicios XMPP (Extensible Messaging and Presence Protocol) (Saint-Andre, Core. 2011). El resto de las publicaciones de Maag que aportan al análisis de este trabajo o similares son (Cavalli, Lallali, Maag, Zaidi, & Morales, 2009) donde menciona, entre otras cosas, que la calidad y el buen desarrollo de casos de pruebas pueden significar el 70% del costo total de un proyecto cuando los test son manuales invitando a la industria e investigadores en hacer los más grandes esfuerzos para automatizar la generación de casos de pruebas automáticas. En (Cavalli, Maag, & Morales, Regression and Performance Testing of an e-Learning Web Application: dotLRN, 2008) muestran la importancia de las pruebas de performance para asegurar la escalabilidad presentando una metodología y utilizan la herramienta OpenSTA, abordan los problemas de regresión, no queda claro si existe una integración de herramientas para detectar y solucionar la regresión sobre los scripts OpenSTA.

## 1.6. Estructura de la tesis

Esta tesis está estructurada de la siguiente manera. El Capítulo I presenta la motivación y un encuadramiento del testing de performance como tópico de investigación dentro de la ingeniería de software y en particular testing de performance para sistemas críticos. Se hace una pequeña descripción de casos de uso reales donde se aplica la metodología exitosamente y se mencionan otros trabajos relevados en la literatura relacionados con el enfoque aquí planteado. El Capítulo II se introduce la metodología utilizada para llevar adelante testing de performance sobre los siguientes casos. A saber, en el Capítulo III consta de aplicar testing de performance en la fusión de dos bancos. El Capítulo IV incluye la aplicación de testing de performance para la relevar el funcionamiento de un *contact center* distribuido. El Capítulo V se estudia y estudia la aplicación de testing de performance para ASSE (Administración de los Servicios de Salud del Estado del Uruguay). El Capítulo VI se hace una discusión de los resultados obtenidos sobre estos sistemas críticos heterogéneos y se presentan conclusiones y algunas recomendaciones a considerar en otros sistemas.

# 2. Metodología

## 2.1. Introducción

En este capítulo se presenta la metodología para pruebas de desempeño utilizada.

La complejidad de las tareas que se deben desarrollar para que una prueba de desempeño sea útil es tal, que merece el tratamiento de dicha actividad como un proyecto en sí mismo. En esta sección se presenta el proceso metodológico seguido para realizar pruebas de desempeño, que puede ser modificado y adaptado según el tipo de prueba de desempeño a realizar. En particular, este capítulo se enfoca en las pruebas de carga.

Se trata de un proceso en etapas secuenciales, sin embargo según el contexto puede ser útil paralelizar parte de las etapas II, III y IV que pueden ser desarrolladas en paralelo. En la Figura 1 se aprecia el diagrama de proceso.

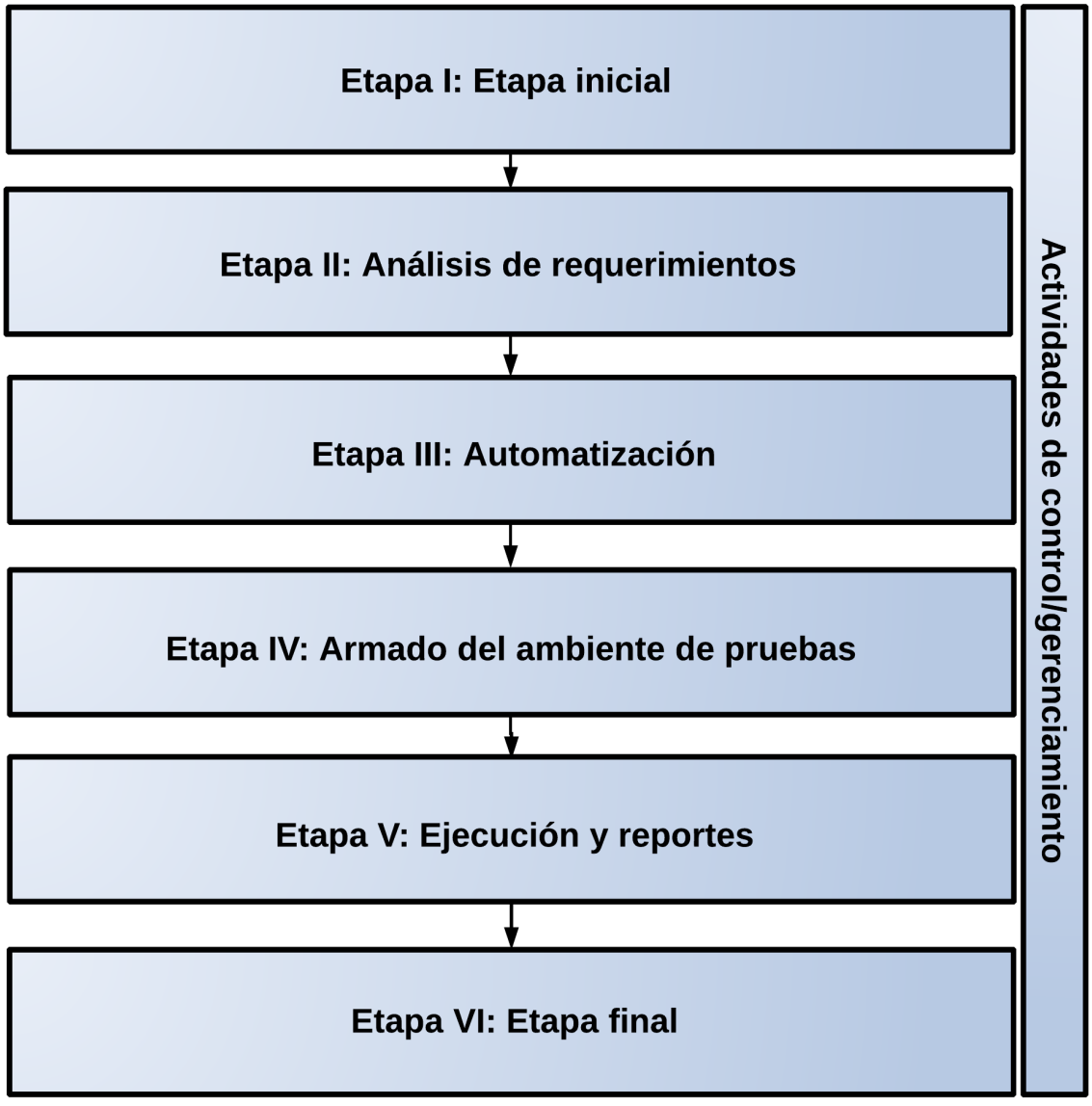



**Figura 1: Etapas de la metodología propuesta de pruebas de desempeño**

Cada una de las etapas de la metodología comprenden una serie de actividades. Las correspondientes a la quinta etapa (Ejecución y reportes) constituyen un ciclo iterativo-incremental. Estas actividades se ven en la Figura 2 y las mismas se describen en las subsecciones siguientes.

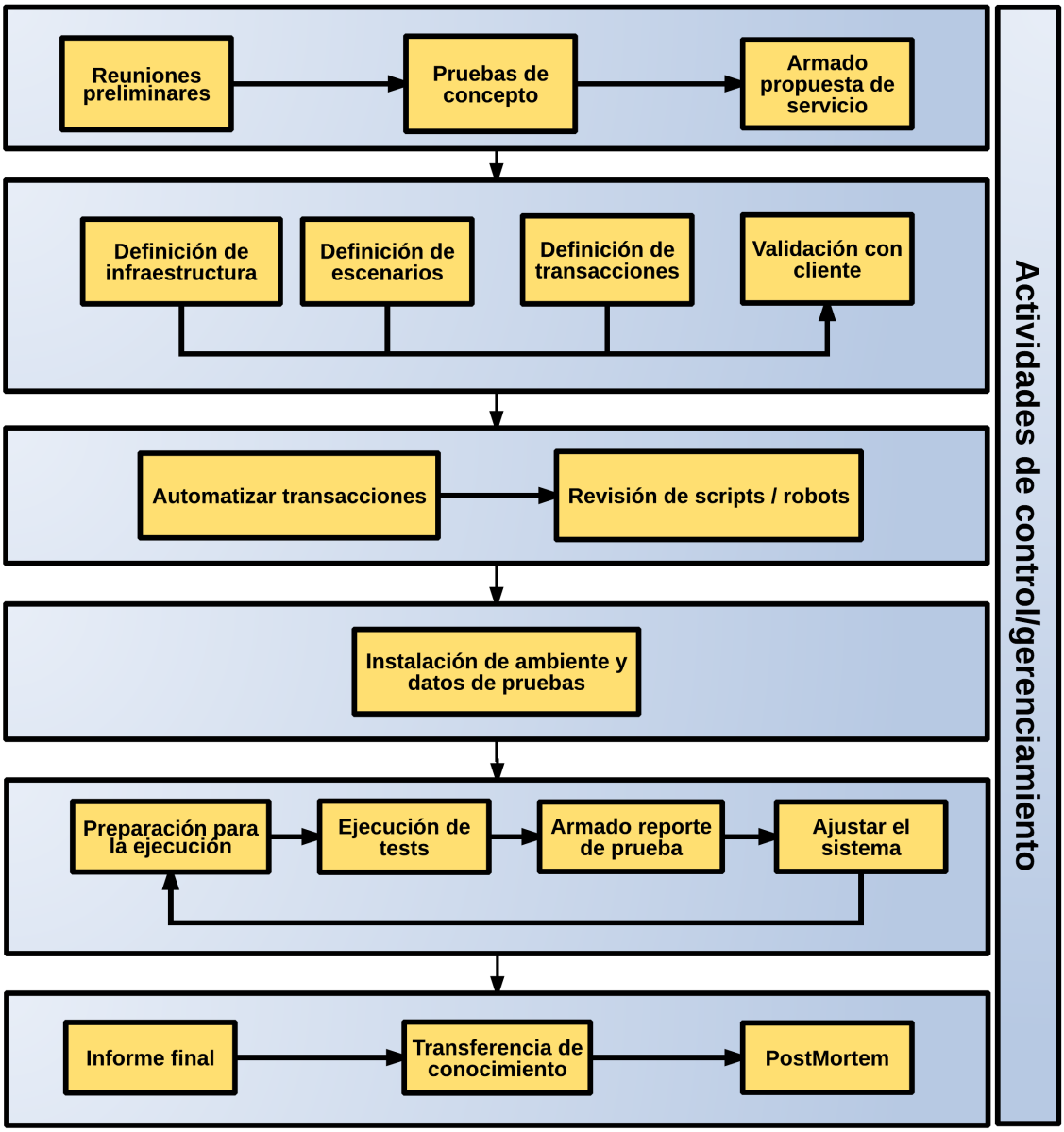



**Figura 2: Actividades por etapas**

La ejecución del proceso de pruebas de desempeño está a cargo de profesionales que cumplen roles bien definidos en el "Anexo II - Tareas y roles". Del lado del equipo de pruebas, existe un líder de proyecto quien se encarga de la gestión del proyecto, e interactúa con su contraparte, por lo general el gerente de tecnología o el líder de proyecto de desarrollo. Además, se cuenta con los *testers* que son los idóneos en las distintas herramientas de *testing* utilizadas en el proyecto, así como también del proceso. En contraparte al equipo de pruebas, se tienen varios roles. Además del líder de proyecto antes nombrado, existen los roles de experto funcional y experto de infraestructura. Los expertos funcionales son personas conocedoras del dominio de la aplicación o del negocio, mientras que los expertos de la infraestructura son personas idóneas en los distintos componentes del sistema. Esto determina una serie de requerimientos que se deben cumplir para que se puedan realizar las actividades definidas en las etapas de la metodología de pruebas de desempeño, en el "Anexo I - Requisitos" se detallan.

## 2.2. Etapa inicial

Esta etapa tiene como objetivo definir el alcance global del proyecto de pruebas, por este motivo al finalizar la etapa todas las partes involucradas deben conocer con detalle qué compromisos tiene cada una para que el proyecto se pueda realizar de manera exitosa.

Aquí se da el primer acercamiento entre las partes. Para definir el proyecto, se realizan reuniones en las cuales se presentan los principales conceptos relacionados a las pruebas de desempeño y se le entrega al cliente un cuestionario básico sobre las características de la aplicación a testear y el contexto bajo el cual se realizarán las pruebas.

Para lograr una mejor estimación del esfuerzo es necesario realizar pruebas de concepto en donde, con el mínimo esfuerzo, se puedan analizar las particularidades del software a probar para, entre los puntos más importantes, definir cuáles son las herramientas más convenientes para las características particulares del proyecto. Generalmente alcanza con establecer y mantener la conexión entre la herramienta y el SUT. Con esto que podría parecer una simple acción entran en juego el protocolo de comunicación y mecanismos de seguridad. Puede que, eventualmente, sea necesario deshabilitar algún mecanismo de seguridad o dar de baja algunas características que finalmente se encontrarán en producción pues hace inviable el proyecto (ej. automatización de un CAPTCHA). En esos casos es necesario evaluar el impacto en la performance. El objetivo de dichas pruebas es mitigar riesgos técnicos, y conocer la dificultad asociada a las distintas actividades del proyecto.

Luego de recopilar la información medular para definir el proyecto se procede a la elaboración de la propuesta de servicio. Dicha propuesta de servicio plasma en un único documento el contexto de las pruebas de desempeño y el alcance de las mismas, estableciendo de manera precisa los compromisos de cada parte involucrada. Entre otros puntos se precisan las personas involucradas y los roles que tendrán en el proyecto, un cronograma de pruebas, un presupuesto y un acuerdo de confidencialidad. Para avanzar en las próximas etapas es necesaria la aprobación de la propuesta de servicio. Es posible que ante ciertos argumentos no prospere o se negocie un alcance distinto.

Cuando un proyecto no puede asegurar alguno de los requisitos o se tienen datos o sospecha sobre posibles problemas de performance, se sugiere comenzar por una etapa de Análisis y diagnóstico. Ésta puede estar enfocada en configuraciones y evaluaciones con herramientas automáticas que brinden rápidamente una pista de problemas gruesos. El objetivo de esa etapa es detectar potenciales causas de mala performance en la configuración del sistema con herramientas genéricas. Se sugiere la elaboración de los siguientes elementos:

- Reporte de hallazgos: memorando que describe el incidente, y es válido para errores funcionales, de concurrencia, seguridad, desempeño, u otro tipo de problemas que quedan visibles. Tiene como objetivo reportar los problemas antes mencionados y, eventualmente, posibles soluciones.
- Reporte de avance: se mencionan los avances de cada jornada y los compromisos pendientes. Tiene como objetivo dar visibilidad al estado del proyecto.

## 2.3. Análisis de requerimientos

Esta etapa comienza luego de concluida la etapa inicial, donde deberá quedar claro el contexto general del proyecto. El objetivo primordial de esta etapa es profundizar la recopilación de datos que se comenzó en la etapa inicial. Los principales tópicos a analizar y definir son los escenarios, las transacciones a incluir en las pruebas, la infraestructura sobre la cual ejecutarán las mismas y los datos a utilizar.

Los escenarios especifican las diferentes condiciones de uso que debe soportar el sistema cuando el mismo se utiliza en producción (este término es conocido como *workload*). El término transacciones se refiere en este contexto a los ciclos funcionales que pueden ejecutar los usuarios del sistema. Definir los escenarios[1] es el punto más complejo de los requerimientos. Generalmente se asocia un escenario a un determinado momento del día, por ejemplo se puede definir un "escenario diurno" como la operativa que determina la carga que sufre el sistema a diario de 13:00 a 14:00 horas. Esa carga prevista se toma como referencia y se la denomina escenario del 100%. Luego, en la etapa "Ejecución y reportes" (Sección 2.6), se ejecutan distintos porcentajes de ese modelo de carga.

Una vez definida la ventana horaria a simular en las pruebas, se deben definir todos los componentes que conforman el escenario. Estos componentes son entre otros las transacciones que se ejecutan en esa ventana horaria, la cantidad de usuarios que ejecuta cada transacción, la cantidad de veces que ejecutan cada transacción, la forma de ingreso al sistema por parte de los usuarios y los procesos en lote (*batch*) que pueden correr en paralelo en dicho horario. Asociado a los escenarios se define la infraestructura sobre la que el sistema será puesto (o se encuentra) en producción. Es medular en las pruebas de desempeño poder definir estos datos de la manera más precisa posible, ya que luego la simulación de la carga se basará justamente en esta definición y la metodología no variará si es más o menos carga la que sufre el sistema. Para poder analizar y definir todos los elementos del escenario, es imprescindible contar con un equipo interdisciplinario con responsables funcionales y desarrolladores así como también basarse en estadísticas, ya sean del mismo sistema que se va a implantar o de alguno anterior que se usara en la operativa a modelar.

Por otra parte, se debe definir un guión preciso para cada transacción donde se incluya cada acción que realiza el usuario para ejecutarla, y la correspondiente respuesta del sistema. Cabe destacar que se debe buscar mantener acotada la cantidad de transacciones que se incluirán en la prueba para poder manejar la complejidad que implica la automatización de las mismas. El criterio para la selección de las transacciones dentro del escenario elegido se basa en un análisis de riesgo, seleccionando aquellas que se ejecutan masivamente sobre el servidor, aquellas que se estima que realicen un consumo elevado de recursos o aquellas que son criticas para el negocio. Otro criterio utilizado para poder reducir la cantidad de transacciones es la simplificación de la realidad. Por ejemplo si la realidad contempla una transacción de inserción y otra de modificación, generalmente se puede incluir en la prueba sólo una (la más critica o riesgosa), y tomar de las estadísticas de ejecución la suma de ambas, considerando que el consumo de recursos de las dos es similar o equivalente. En este caso, se busca hacer la prueba realizable, aún cuando el modelo que se ejecute no sea 100% fidedigno con la realidad.

En cuanto a la infraestructura se debe precisar la versión de cada uno de sus componentes lógicos y físicos (software de base y hardware) así como también recopilar los indicadores de a medir, los cuales se detallan en la etapa "Armado del ambiente de pruebas" (Sección 2.5). Es importante recolectar las métricas definidas para los diferentes componentes y contar con expertos que, en función de los resultados obtenidos en las sucesivas ejecuciones, sean capaces de sugerir mejoras y realizar la "sintonía" o "tuning" de la configuración. Dentro de la infraestructura definida deben considerarse todos aquellos elementos que se vayan a ejecutar en producción concurrentemente con el escenario.

Otro punto importante es definir los datos que se utilizaran para ejecutar cada transacción del escenario de carga, ya que los mismos son una condición imprescindible para comenzar con la ejecución. Se definen tanto los datos de la base de datos, como los datos que se usarán como entrada para las transacciones a ejecutar. Es necesario que los datos sean similares a los que se

[1] Un novedoso modelo de carga se investigó en el marco del proyecto "Modelado y ejecución de pruebas de performance" (Barbatto & Díaz, 2014).

espera tener en producción, tanto en calidad como en cantidad; de no ser así no se estaría modelando adecuadamente la realidad y muchos de los potenciales incidentes pueden no llegar a observarse en las pruebas. Estos datos pueden surgir de migraciones de sistemas anteriores, o pueden ser creados manual o automáticamente. También se pueden utilizar los artefactos creados en la etapa de "Automatización" (Sección 2.4) para asistir a la tarea de creación de datos. A menudo la creación de datos históricos, las pruebas de sistemas basados en flujos de trabajos (*workflow*s) o la conversión de datos actuales a datos anónimos (para no exponer información de acceso restringido) pueden hacer que la creación de datos demande muchos recursos. Estos recursos generalmente no son del equipo de pruebas, sino del equipo de desarrollo o funcional, debido, principalmente, al conocimiento que los mismos tienen del esquema de datos sobre el que se basa el sistema. La tarea de seleccionar los datos a utilizar como entrada de las transacciones es también critica, ya que la variedad que entre ellos exista debe reproducir la realidad. Una inapropiada mezcla de datos de entrada puede arruinar una prueba de desempeño.

Finalmente, es deseable establecer un criterio de aceptación u objetivo a superar. Este puede ser definido en base a una cantidad de operaciones a realizar en una ventana de tiempo, al consumo de recursos del sistema o al tiempo de respuesta de cada transacción, cualquiera de ellos medido en la situación de carga del sistema definida por el escenario correspondiente. Este criterio de aceptación es el principal objetivo a ser alcanzado y, ante todo, el principal elemento que define cuándo es aceptable dejar de ejecutar las pruebas.

Esta etapa se da por finalizada cuando se validan los datos definidos con todas las partes involucradas en el proyecto.

## 2.4. Automatización

La etapa de automatización tiene como objetivo construir los artefactos necesarios para poder reproducir de manera automática el escenario de carga definido en la etapa "Análisis de requerimientos" (Sección 2.3). Esto generalmente incluye los scripts o robots que oficiarán de usuarios virtuales y los datos de entrada para los mismos.

Los principales artefactos a construir son los scripts o robots capaces de ejecutar las transacciones, siguiendo el guión definido en la etapa anterior. Estos scripts, reproducidos por los denominados "usuarios virtuales", son capaces de ejecutar una transacción logrando que el servidor no distinga si la misma es ejecutada por un usuario real o un usuario virtual.

Una son prueba de performance automatizada bien configurada y ejecutada tiene varios beneficios, entre ellos: la repetitividad de su ejecución, facilita su análisis, brinda tiempos de respuesta objetivos, se puede ejecutar de forma desatendida y ahorra tiempo y esfuerzo de RRHH y uso de infraestructura. En contrapartida requiere del uso de herramientas de automatización y el esfuerzo de desarrollo (automatización) para reproducir los escenarios definidos. Existen varias herramientas que permiten la creación de los scripts y luego la ejecución de manera eficiente de cientos de usuarios virtuales por máquina. Esas herramientas proveen la capacidad de ingresar los distintos parámetros del escenario definido en la etapa anterior, como son la cantidad de usuarios que ejecuta cada transacción, la cantidad de veces que ejecuta cada usuario, la forma de ingreso de los usuarios al sistema (cadencia, *ramp-up*), el tiempo en el que se ejecuta el escenario, entre otros. Además permiten la monitorización de la carga (por ejemplo la cantidad de usuarios activos) y de los tiempos de respuesta de cada transacción. Por otro lado permiten distribuir los usuarios virtuales en varias máquinas y algunas tienen incorporadas componentes de monitorización de infraestructura.

Muchas veces, debido a los distintos protocolos de comunicación que un sistema puede utilizar entre el cliente y el servidor, no existe una única herramienta que pueda realizar las actividades con todas las transacciones que conforman el escenario y menos aún en los diferentes proyectos de pruebas de desempeño. En estos casos se torna imprescindible desarrollar herramientas o

integrar varias de ellas que permitan la automatización de cada transacción. La selección de la, o las, herramientas adecuadas puede ser de vital importancia en el costo/beneficio del proyecto. Ciertas herramientas, categorizadas como "worldclass", pueden resolver muchos de los problemas más difíciles, relacionados con protocolos e incluso hacer muy sencilla la tarea de producción de los scripts, pero su costo puede hacer inviable el proyecto. En el otro extremo, herramientas "open source", de distribución gratuita, pueden demandar mayor esfuerzo de programación e incluso no ser capaces de resolver la comunicación. En caso de que no se encuentre una herramienta adecuada para alguna transacción, siempre se puede optar por ejecutarla manualmente, aunque hay que tener en cuenta las implicancias que esto conlleva, como son: la dificultad para escalar, la subjetividad del usuario que lo ejecute (por más que mida con un reloj los tiempos de respuesta) y la pérdida de la repetitividad de la prueba (en este caso por la incapacidad de reproducir exactamente la misma prueba debido, al menos, a variaciones en los tiempos humanos).

Una fase importante dentro de la metodología es la revisión de los scripts/robots, más allá de la herramienta seleccionada, es fundamental tener en cuenta que este *testware* juega un papel protagónico en las pruebas de desempeño y un error puede llevar a conclusiones equivocadas. Por este motivo y debido a que el *testware* es software, el proceso propone la realización de revisiones estáticas del código de los scripts (asistidas con listas de chequeo) así como pruebas unitarias de los mismos. También es recomendable seguir normas de código y nomenclatura de archivos para las distintas herramientas.

Es importante destacar que no tiene sentido hacer una prueba de rendimiento de una aplicación que no funciona. Es altamente recomendable tener una versión "congelada" sobre la cual realizar las pruebas, ya que "pequeños" cambios pueden impactar en los scripts, obligando a re-programarlos teniendo que rehacer parte del trabajo realizado en esta etapa. Por ejemplo, un script que envía una petición HTTP hacia un servidor web que incluye 5 parámetros (GET) y debido a los cambios en la aplicación necesita 10 parámetros, hace necesario generar nuevamente el script porque puede que no contemplé la carga actual y seguramente el falle (el servidor notará que hay parámetros que faltan). Particularmente las aplicaciones desarrolladas con lenguajes de cuarta generación, por sus características, son las que se tiene menos control de los cambios que se generan y afectan las automatizaciones.

Al margen de los objetivos de la etapa de automatización, en ocasiones se detectan problemas o posibles mejoras al sistema bajo prueba durante la misma. Si hay algún detalle importante para reportar, incluso si no es objetivo directo de las pruebas que se están realizando, se reporta bajo la categoría de hallazgo (*Finding*).

### 2.4.1. Herramientas de generación de carga

En esta sección se muestra un análisis e investigación sobre las principales herramientas existentes para la automatización.

Las herramientas se utilizan para robotizar el uso de una aplicación y generalmente proveen un lenguaje para construir los scripts que terminarán simulando los usuarios virtuales.

Este análisis puede servir para la mayoría de las herramientas, sin embargo se agregan algunas particularidades para esta etapa.

Existen herramientas libres (*Open Source* y *freeware*) y herramientas comerciales. La elección de una u otra dependerá del protocolo en el cual se base la aplicación, la experiencia sobre la herramienta, el contexto en el cual estemos trabajando, el dinero que pensemos invertir en esta etapa, entre otros.

Generalmente las herramientas tienen un módulo grabador, que permite capturar una interacción de un usuario real con el servidor del protocolo en cuestión. Una interfaz de usuario

para adaptar el script. Adaptar el script implica la presencia de un lenguaje de programación (en algunos casos un lenguaje visual).

Previo a la ejecución de las pruebas es necesario probar cada script. Si bien, en general, se podrá interceptar el tráfico (con algún *sniffer*) y verificar que el generado por cada script es similar al generado por un usuario real al interactuar con la aplicación, la capacidad de depurarlo es un aspecto útil.

En su mayoría estas herramientas se pueden denominar de generación de carga pues cuentan con una interfaz para ejecución de la prueba donde el *tester* podrá ver determinados resultados (como el tiempo de respuesta) en tiempo real. Algunas cuentan con una interfaz para el análisis de resultados e incluso facilitan la recolección de datos de agentes en los servidores del SUT. Para alcanzar determinados niveles de carga puede ser necesario distribuir la carga entre distintos equipos que generarán la carga, es por eso que un punto a evaluar será la facilidad que brinde para distribuir la carga. Se debe evaluar si soporta generación distribuida de carga y conocer la arquitectura y protocolos.

Durante la ejecución y a posteriori será necesario analizar los resultados de la prueba. Es importante saber con qué resultados se puede contar durante la ejecución (es decir en tiempo real) y luego. Las gráficas son más útiles que los *logs* pero muchas veces pueden consumir muchos recursos y hacer inviable su uso, al menos, durante la ejecución de la prueba. Más allá del consumo de recursos también será necesario evaluar si son entendibles y útiles las gráficas que genere la herramienta, muchas veces no lo son y es necesario exportar los datos y generar algún script o utilizar una herramienta externa para su procesamiento.

Las funcionalidades que aporten valor a la monitorización pueden ser otro aspecto a evaluar. Dependerá de lo que se cuente y las necesidades. A modo de ejemplo si la herramienta se puede integrar y correlacionar datos que brinde otra herramienta. En caso de poder integrar la monitorización también interesa conocer que variables se pueden monitorizar, el formato en que se persisten y los protocolos o aplicaciones se utilizan para realizarla ej.: SNMP, NTPerformance o comandos Unix. En la siguiente sección (2.5 - “Armado del ambiente de pruebas”) se desarrollan aspectos relacionadas con la monitorización.

Hay otras funcionalidades que pueden ser útiles y facilitan algunos aspectos en la emulación como lo son *IP Switching*, simulación de distintos anchos de banda, la posibilidad de realizar otros tipos de prueba (como por ejemplo funcionales).

Más allá de las funcionalidades hay dos aspectos que vale la pena evaluar: uno es la documentación y el otro es el soporte. El nivel de la documentación técnica, el manual de usuario y qué tan actualizada se encuentra pueden son importantes así como los foro para realizar consultas y la calidad de las respuestas en lo que respecta a tiempo y forma.

Hay veces que los protocolos del SUT no son conocidos y es necesario desarrollar o adaptar una herramienta es por eso que interesa también, conocer las posibilidades y limitaciones de extensión de la herramienta así como los mecanismos que ofrece son importantes conocerlos. En el caso de que sea de código abierto (*open source*) el lenguaje en el cual esta implementada y la documentación técnica existente serán de los puntos más importantes. En el caso de que no provea el código se debe saber que mecanismos de extensión se tienen, por ejemplo: *plugins*, test especiales, entre otros.

La siguiente tabla muestra las principales herramientas y algunas características a destacar.

| Herramienta | Licencia | Soporte | Protocolos | Plataformas |
|---|---|---|---|---|
| HP LoadRunner | Paga | Completo | Múltiples | Windows y Linux (solo generación de carga) |
| IBM Rational Performance Tester | Paga | Completo | Múltiples | Múltiples |
| QALoad | Paga | Discontinuado | Múltiples | Múltiples |
| Microfocus Silk Performer | Paga | Completo | Múltiples | Múltiples |
| OpenSTA | GNU GPL | Discontinuado | HTTP, HTTPS | Windows/C++ |
| JMeter | APACHE LICENSE | Comunidad activa | Múltiples | Java |
| SIPp (HP) | GNU GPL | Actualizaciones | SIP | Java |
| ISOLoadGenerator | Propietaria (CES) | Actualizaciones | ISO8583 | Java |
| SoapUI | (1) Open source / (2) Pago | Actualizaciones | Múltiples | Múltiples |
| LoadUI | (1) Open source / (2) Pago | Actualizaciones | Múltiples | Múltiples |

**Tabla 1 - Principales herramientas de generación de carga**

HP LoadRunner es la herramienta de pruebas de Hewlett-Packard. LoadRunner era de Mercury Interactive hasta que HP adquirió la compañía en noviembre de 2006. LoadRunner tiene más de una década y media y logra simular la actividad del usuario mediante la generación de mensajes o mediante la simulación de la interacción con la interfaz de usuario, tales como pulsaciones de teclas o movimientos del ratón. Soporta diferentes aplicaciones y tecnologías (Microsoft .NET y Java, servidores de bases de datos como Microsoft SQL Server y Oracle, protocolos como DNS, FTP, LDAP, IMAP, MAPI, POP3 y SMTP y

tecnologías de cliente remotos como Citrix ICA y RDP). Desde 2010 se trabaja para brindar la herramienta como servicio, inicialmente se desplegó en la nube de Amazon Elastic Compute Cloud, luego en Microsoft Azure y recientemente en Google Compute Engine.

Rational Performance Tester es una herramienta de prueba de rendimiento basado en Eclipse que tiene más de una década. Pertenece a la suite de herramientas de IBM y permite realizar pruebas de rendimiento automatizado.

QALoad es una herramienta desarrollada por Compuware discontinuada pero vigente, hoy pertenece a Micro Focus dado que en 2009 adquirió el área “Compuware Testing and ASQ Business” por 80 millones de dólares (Microfocus, 2009). Micro Focus también es dueña de Silk Performer que brinda soporte a más tecnologías que QALoad y se sugiere como el producto sustituto. Silk Performer Permite probar la performance de aplicaciones web, móviles y empresariales. Fue desarrollado originalmente por Segue Software que fue adquirida por Borland en 2006. Borland fue adquirida por Micro Focus en 2009.

OpenSTA es una opción libre para generar scripts, cuenta con un lenguaje de programación compilado (SCL[2]) para definir el comportamiento de los usuarios virtuales. Se puede realizar una prueba de carga para sistemas accesibles vía HTTP y HTTPS. OpenSTA fue desarrollado originalmente por Cyrano. Si bien parece descartable, pues está discontinuado y la versión más reciente es la 1.4.4 del 27 de octubre 2007, OpenSTA se utiliza para algunas pruebas dado que es una herramienta muy eficiente en cuanto a la cantidad de usuarios activos que puede generar en relación al uso de recursos (*hardware*) así como la velocidad y utilidad de los reportes que se pueden crear.

Apache JMeter es una aplicación de escritorio desarrollada en Java, diseñada para generar carga y medir la performance del SUT. Originalmente fue diseñada para aplicaciones web pero se fueron agregando capacidades y extensiones soportando en la actualidad varios protocolos y tecnologías como HTTP, HTTPS, SOAP, REST, FTP, bases de datos vía JDBC, LDAP, MOM vía JMS, SMTP(S), POP3(S), IMAP(S) entre otros. El proyecto tiene varios contribuyentes hay seguimiento de los *bugs* y varios liberaciones de versiones estables por año. El resultado de la grabación de tráfico es la generación de una estructura arborescente donde cada nodo puede ser editado y configurado, esa estructura no es más que un lenguaje de programación es visual que representa la traza que describe la comunicación entre el servidor y el usuario. Al guardar el plan genera un archivo XML que JMeter es capaz de interpretar y generar la estructura mencionada.

SIPp (Hewlett-Packard, 2004) es una herramienta de código abierto que permite probar y generar tráfico con el protocolo SIP. Hewlett-Packard es quien la liberó, incluye escenarios básicos en los que agentes realizan llamadas. Muestra dinámicamente las estadísticas de las pruebas que se van ejecutando. Se puede utilizar para probar SIP *proxies*, B2BUAs y SIP media servers (ej. centrales telefónicas basadas en este protocolo). SIPp puede generar las trazas que se dan entre el cliente y el SIP core.

SOAPUI / LoadUI son frameworks para pruebas de web services (funcional / desempeño). SOAPUI es la herramientas diseñada principalmente para testing funcional de APIs SOAP y REST, se incluye en la lista porque tiene la posibilidad de dar soporte a pruebas de carga con las tecnologías comunes de la industria. LoadUI facilita la generación de pruebas de carga aprovechando el testware generado con SOAPUI. Se distribuyen versiones gratis, de código abierto y pagas (que se pueden probar gratis durante 14 días). Las mayores diferencias entre las versiones radican en los reportes y las pruebas distribuidas. Hay versiones recientes y en *github* se puede acceder al código y generar *branches* (o *forks*), actualmente existen proyectos paralelos y contribuciones sobre este proyecto.

Como se ve en la tabla anterior para replicar el tráfico que se genera entre el cliente y el SUT hay varias herramientas que son aptas, sin embargo puede ser necesario utilizar una combinación de las mismas, extender alguna o desarrollar una nueva. Las herramientas de código abierto antes mencionadas las hemos extendido y mejorado, a la vez que otras no tan populares. En el sector financiero el protocolo ISO8583 es ampliamente difundido y si bien se pueden encontrar diversas herramientas que implementan este protocolo, permitiendo enviar mensajes mediante el mismo hacia un "Switch Financiero" (problema también abordado en el CES), ninguna de ellas cumple el requerimiento fundamental de toda prueba de performance: permitir la generación automática de carga masiva que logre emularlo. Este protocolo es utilizado para el intercambio de información financiera, principalmente por ATMs (*Automatic Transaction Machine*) y POS (*Point of Sale*). Es decir cuando realizamos un retiro de un cajero automático o una compra en un local de venta utilizando una tarjeta de crédito/débito, estamos estableciendo una comunicación vía ISO8583 con una institución financiera, cuyo sistema de administración de tarjetas se encargará de procesar la petición recibida y enviar una respuesta indicando el resultado de la transacción. El componente que procesa la información, típicamente se llama "Switch Financiero". Éste es un

[2] Script Control Language

caso en que fue necesario desarrollar una herramienta generadora de carga, esta herramienta ha permitido participar en otros proyectos. Fue necesario entender la especificación del estándar y desarrollar la aplicación acorde al mismo, que hoy en día luego de varías mejoras es una herramienta estable capaz de soportar hasta 200 transacciones por segundo en un simple PC de escritorio; cifra que ha permitido emular los casos que se han presentado (en caso de necesitar más siempre se puede distribuir la carga). El contar con esta herramienta, y poder usarla en conjunto con otras para poder simular todos los puntos de entrada de un *core* financiero, ha permitido llevar adelante exitosamente múltiples proyectos en los cuales ha estado involucrado el CES, logrando generar la carga requerida para poder cumplir los objetivos primordiales de una prueba de performance.

Si bien los herramientas pagas pueden ser más potentes y de fácil uso, la relación costo/beneficio hace que el uso de las herramienta libres puedan catalogarse como buenas decisiones. Algunas de las decisiones por las que se utiliza herramientas libres en estos proyecto son:

- Generalización de scripts.
- Sin costo de licenciamiento.
- Conocimiento en la interpretación de los *logs* y generación de informes.
- Experiencia de uso en proyectos previos por parte del equipo de pruebas.
- Posibilidad de escalar la carga.

Particularmente en el caso de OpenSTA y JMeter la grabación y reproducción a nivel del protocolo de comunicación (manejo de tráfico HTTP) es lo que permitió usarlas en sistemas web, mientras que en el caso de la simulación del tráfico SIP, correspondiente a las llamadas telefónicas, tanto sea de agentes como clientes la herramienta fue SIPp.

## 2.5. Armado del ambiente de pruebas

Esta etapa tiene como objetivo dejar lista toda la infraestructura de pruebas para la ejecución de las mismas, compuesta por todo el ambiente de pruebas, o sea el hardware y software que soportan el sistema bajo prueba y de las generadoras de carga, las conexiones red y los datos de prueba. Si bien se recomienda realizar esta etapa previo a la automatización, debido a que en la etapa de análisis de requerimientos ya se cuenta con la información de la infraestructura, generalmente tiene costos asociados que se minimizan al realizarla justo antes de la ejecución. Los costos asociados pueden ser licencias de software o disponibilidad del equipo de producción (puede que el mismo se tenga que comprar, alquilar o utilizar para otras actividades).

Dentro del software a instalar, se encuentran las herramientas de monitorización. Monitorizar el comportamiento de la infraestructura y del sistema en general es de primordial importancia en una prueba de desempeño. Estudiar el comportamiento de los servidores, la red e incluso de los propios generadores de carga es de relevancia para saber el porqué del comportamiento de los diferentes componentes de la aplicación. Por ello, además de la generación de la carga, una prueba de desempeño tiene su foco en la monitorización. En esta etapa deben quedar definidas y configuradas las herramientas a ser utilizadas para monitorizar. Entre los indicadores a recolectar durante la prueba, se pueden distinguir distintos niveles. Un primer nivel generalmente se compone de indicadores que se van a monitorizar también cuando el sistema esté en producción, y por tal motivo, son poco intrusivos, es decir, su recolección generan poco esfuerzo extra al sistema y su incidencia en los tiempos de respuesta obtenidos de las transacciones es despreciable. Estos indicadores que se van a tomar en producción deben estar activos durante la ejecución de las pruebas. En caso que no se prevea monitorizar al sistema en producción, o que la monitorización prevista no brinde la información mínima necesaria para apreciar el desempeño del sistema, las herramientas que se decida usar deberán ser lo mínimo intrusivas posible, para no alterar el modelo de carga. En estos casos, es recomendable medir la incidencia de la herramienta de monitorización durante la etapa de ejecución de la línea de base. Por otra parte

un siguiente (segundo) nivel se obtienen de herramientas que generalmente introducen un impacto relevante en el desempeño del sistema y se utilizan para profundizar sobre un problema en particular a la luz de la información arrojada por la monitorización de primer nivel. Como es de suponer, cuando se ejecutan pruebas en las que se tienen activados estos indicadores, los tiempos de respuesta no son tenidos en cuenta, teniendo la prueba su foco en la solución o detección de datos para la corrección de problemas o posibilidades de mejora.

Es importante recordar que las generadoras de carga y la red que conecta dichas máquinas con el sistema bajo prueba, también deben estar monitorizadas. Si se desea observar el comportamiento bajo condiciones de carga de una pieza de hardware o software en particular, el eventual cuello de botella debe ser dicho recurso. Esto significa que no se debe introducir otros elementos en la prueba que sean el cuello de botella, en particular los mismos clientes que generan la carga.

### 2.5.1. Herramientas de monitorización

En esta sección se muestra un análisis e investigación sobre las herramienta utilizadas para la monitorización.

El primer punto a tomar en cuenta para elegir una herramienta es lo que desea monitorizar. Monitorizar todos los contadores de los equipos no solo puede degradar el sistema por su intrusividad sino que podría llevar mucho tiempo su revisión y no aportar, por lo tanto es importante detectar y acordar los indicadores a revisar.

Los equipos que forman parte de la infraestructura del SUT, pueden ser potenciales cuellos de botella y por lo tanto deberá evaluarse los contadores a monitorizar. En general los sistemas operativos cuentan con alguna herramienta que permite monitorizar, por ejemplo: CPU, memoria, red y disco. Tal es el caso de *Performance Monitor* (perfmon) incluida por defecto en todos los sistemas Windows NT.

En la medida que se agreguen componentes especializados se puede discriminar el CPU total del dedicado a determinados procesos, por ejemplo a base de datos, el paginado de usuario, el uso físico del disco, CPU de disco, uso de LAN.

Al agregar entornos de ejecución, por ejemplo máquinas virtuales de Java o .NET existen tecnologías que permiten monitorizar el comportamiento dentro de la máquina virtual. En el caso de Java, JMX (*Java Management Extensions*) es la tecnología que permite exponer información dentro de la JVM (*Java Virtual Machine*) y por lo tanto su monitorización. En .NET, WMI (*Windows Management Instrumentation*) es la tecnología equivalente. En el caso de servidores de aplicación que dan soporte a estas tecnologías como ser WAS (*WebSphere Application Server*), para el caso de Java, se puede monitorización el comportamiento de la JVM. El uso de CPU y memoria (ej. tamaño del *heap* y en particular cada zona así como el momento y los tiempos del *garbage collections*) suelen ser un punto útil a configurar y analizar.

Otro punto a considerar es la actividad en la base de datos. Conocer las SQLs más referenciadas y los mayores tiempos de ejecución de las sentencias así como los *locks* suelen dar pistas de posibles cuellos de botella o oportunidades de mejora.

Para la monitorización del ancho de banda se pueden utilizar herramientas del sistema operativo o herramientas particulares. Comúnmente es útiles conocer, por ejemplo, el ancho de banda utilizado o necesario con el centro de cómputos.

La siguiente tabla muestra las principales herramientas y algunas características a destacar. Se diferencias dos grandes grupos de herramientas: las que miden los datos (los monitores) y las que los centralizan, procesan y/o presentan.

| Herramienta | Licencia | Tecnología | Componente | Recurso | Tipo |
|---|---|---|---|---|---|
| nmon | Libre | Basados en Unix | Sistemas Operativos | CPU, Disco, Memoria, entre otros. | Mide Centraliza Presenta |
| nmon Analyzer | Libre | Multiplataforma | Sistemas Operativos | nmon | Presenta |
| Performance Monitor (Perfmon) | Incluida en sistemas operativos de Microsoft | Basados en Windows NT | Tecnologías Microsoft | CPU, Disco, Memoria, entre otros. | Mide Centraliza Presenta |
| Slow Query log | Incluida en MySQL | Multiplataforma | MySQL | SQLs | Mide |
| MySQLReport | Incluido en MySQL | Multiplataforma | MySQL | Actividad base datos | Presenta |
| stats_* y Statistics Collector | Incluido en PostgreSQL | Multiplataforma | PostgreSQL | Actividad base datos | Mide |
| SQL Query analyzer | Incluido en SQL Server | Basados en Windows NT | SQL Server | SQLs | Mide Presenta |
| Statspack | Paga | Multiplataforma | Oracle | Actividad base datos | Mide Procesa Presenta |
| SQL Trace | Paga | Multiplataforma | Oracle | SQLs | Mide |
| NetFlow Analyzer | Freeware con límites | Basados en Windows NT | Red | Ancho de banda | Mide |
| JConsole | Incluida en Oracle JDK | Multiplataforma | JMX | JVM (Heap, Threads, entre otros) | Mide Centraliza Presenta |
| Cacti | GNU GPL | Multiplataforma | Sistemas Operativos | CPU, Red, Memoria, entre otros. | Mide Centraliza Presenta |
| RRDTool | OpenSource | Multiplataforma | Gráficas | CPU, Red, Memoria, entre otros. | Presenta |
| GCViewer | LGPL | Java | Java Garbage Collection | JVM (Heap, Threads, entre otros) | Presenta |

**Tabla 2 - Principales herramientas de monitoreo**

Nmon es una herramienta de monitoreo gratuita para analizar la performance de sistemas operativos basados en Unix. En AIX viene instalada por defecto y es posible instalarla en Linux. Colecta variables nativas de sistema operativo que se pueden acceder desde diversas formas siendo útil su registro en un archivo de texto. Nmon Analyzer es una macro Excel que puede tomar estos archivos y generar reportes. Fue desarrollada por IBM pero no brinda oficialmente soporte.

Performance Monitor es un programa de monitoreo de la familia Windows NT su predecesor fue System Monitor (desde Windows 95 hasta ME). Se utiliza para controlar diversas actividades en

un equipo local o remoto mediante la medición del rendimiento del hardware (como el uso de la CPU o el uso de memoria), servicios y aplicaciones. Tiene más de 350 contadores disponibles y puede mostrar información de forma gráfica y alterar intervalos de tiempo. Las categorías que se puede monitorear pueden crecer de acuerdo a los servicios instalados en el sistema.

Slow query log brinda las sentencias SQL que tuvieron más de determinado tiempo en ejecución (long_query_time en segundos) y requieren por lo menos determinada cantidad de filas para ser examinadas (min_examined_row_limit).

MySQLReport es un script en Perl que muestra un informe variables de estado de MySQL (tomado de la salida de SHOW STATUS de MySQL) que puede ayudar a entender lo que está sucediendo dentro de MySQL y diagnosticar problemas.

Statistics Collector es un subsistema que ayuda la recopilación y presentación de información sobre la actividad del PostgreSQL por ejemplo los accesos a las tablas e índices. También soporta notificaciones.

SQL Query analyzer es una herramienta gráfica que se utiliza para calcular el tiempo de ejecución de las consultas, store procedures de depuración (T-SQL debugger), depurar los problemas de rendimiento de las consultas revisando: planes de ejecución, trace, estadísticas y mediante el asistente para optimización de índices.

Statspack es un conjunto de scripts SQL, PL/SQL y SQL*Plus que permiten la recolección, automatización, almacenamiento y visualización de datos de rendimiento del motor de base de datos Oracle. Almacena las estadísticas de rendimiento en las tablas de Oracle, que luego pueden ser utilizados para análisis y la presentación de informes. Los datos recogidos se pueden analizar utilizando Statspack reports, que genera una página web que resume la salud y la carga, las sentencias SQL que consumen más recursos entre otros.

SQL Trace proporciona información sobre el rendimiento de las sentencias SQL de Oracle: Tiempos de CPU, lectura física y lógica, número de filas procesadas, *misses* en la caché, nombre de usuario, commit y rollback. Se puede activar para una sesión o de una instancia.

NetFlow Analyzer es una herramienta de análisis de tráfico y monitoreo de ancho de banda, proporciona visibilidad en tiempo real sobre el rendimiento del ancho de banda de la red, quién está utilizando y para qué. También es útil para hacer análisis forense de redes.

JConsole es una herramienta que permite monitorear JVMs en un equipo local o remoto. Está incluido en la JDK (*Java Development Kit*) y proporciona información sobre el consumo de recursos y rendimiento de las aplicaciones que se ejecutan en la plataforma Java, dentro de la JVM. Utiliza la tecnología JMX.

Cacti es una herramienta que presenta gráficamente en una web el resultado de monitorear elementos de la red, tales como carga de la CPU y la utilización del ancho de banda de la red de un equipo, que registran datos en la herramienta RRDtool. Un uso común es monitorear el tráfico de red consultando vía SNMP (Simple Network Management Protocol) un *router* o *switch*. Una extensión que hemos implementado es evolución de la experiencia de usuario, mediante la escritura de un log de tiempos de respuesta de una transacción, en este caso se utiliza un script que puede generar carga pero con el fin de tener lo que denominamos un usuario testigo automatizado. Esto brinda una visión objetiva y desatendida de la experiencia de usuario en lo que respecta a performance del sistema desde determinada ubicación.

GCViewer es una herramienta de código abierto para la visualización gráfica de datos producidos por las opciones de Java VM que exponen información sobre el Garbage Collector (-verbose: gc y –Xloggc). También calcula métricas de rendimiento asociadas (rendimiento, pausas acumulados, pausa más larga, entre otros). Es muy útil para ajustar y entender el uso de memoria y la performance del garbage collector asi como su configuración ideal variando tamaños de las distintas zonas.

Hay otras herramientas que son útiles para monitorear algo en particular, una vez que detecta determinado problema y se puede estudiar aislado, generalmente se pueden utilizar para mejorar la performance desde el desarrollo. También facilitan el proceso de automatización, pues permiten revisar las solicitudes y las respuestas (*sniffers*). Es el caso de wireshark, fiddler, profilers (ej. JProfiler y VisualVM), webscarab actual ZAP, YSlow, Page Speed, entre otros.

| Herramienta | Licencia | Tecnología | Componente | Recurso | Tipo |
|---|---|---|---|---|---|
| Firebug | Libre (BSD License) | Multiplataforma | Navegadores | Solicitudes y descargas | Mide<br>Centraliza<br>Presenta |
| Fiddler | Freeware | Basados en Windows NT | Navegadores | Solicitudes y descargas | Mide<br>Centraliza<br>Presenta |
| JProfiler | Paga | Multiplataforma | Java Profiler | JVM (Heap, Threads, CPU, Red, Memoria, entre otros) | Mide<br>Centraliza<br>Presenta |
| VisualVM | OpenSource | Multiplataforma | Java Profiler | JVM (Heap, Threads, CPU, Red, Memoria, entre otros) | Mide<br>Centraliza<br>Presenta |

JProfiler es un profiler para Java, permite identificar los cuellos de botella de rendimiento, pérdidas de memoria, analizar volcados de memoria (*heap dumps*) y el manejo multi-hilos de una aplicación dado que permite el monitoreo de la memoria, consumo de CPU, hilos, memoria sin liberar, entre muchos otros. Se puede conectar a JVMs locales como remotas. Brinda soporte para una integración rápida para distintos tipos de servidores: Apache Geronimo, Apache Tomcat, JBoss, Jetty, entre otros, para otros casos se puede hacer una configuración ad hoc.

VisualVM es una herramienta de *profiling* para la plataforma Java de código abierto. Sus repositorios muestran actividad reciente. Se conecta tanto a JVM locales como remotas.

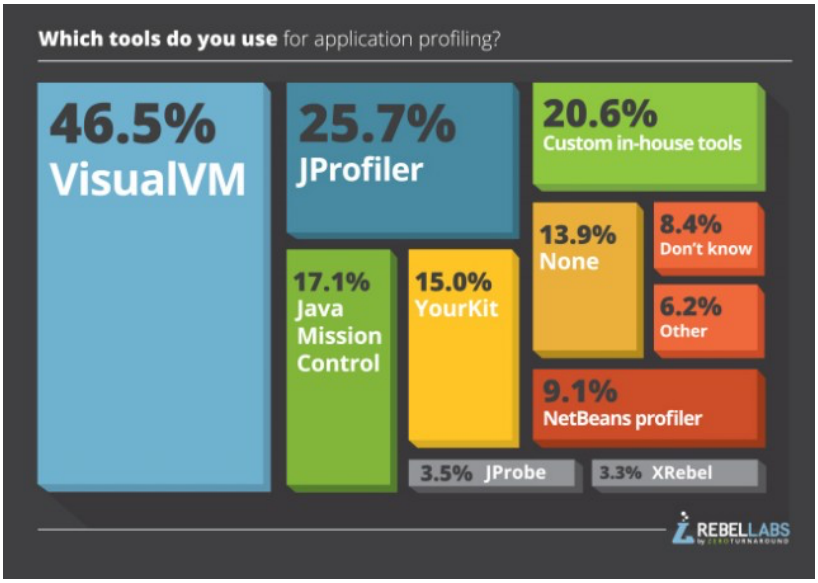



**Figura 3: Market-share Java Profilers**

La figura anterior es del informe *Top 5 Java Profilers Revealed: Real world data with VisualVM, JProfiler, Java Mission Control, YourKit and Custom tooling.*

## 2.6. Ejecución y reportes

Completadas las etapas anteriores, se tienen todos los artefactos necesarios para ejecutar las pruebas de desempeño. Si bien al final de esta etapa se deben haber ejecutado los escenarios previstos en la etapa "Análisis de requerimientos" (Sección 2.3), la metodología propone realizar varias ejecuciones en las cuales la carga sobre el sistema se va incrementando de manera gradual hasta llegar a los escenarios objetivos o superarlos.

El punto de partida de las pruebas de performance, propiamente dichas, es decir previo a la ejecución de los escenarios, es la ejecución de los denominados *baselines* (línea de base). Cada transacción seleccionada es ejecutada de manera individual y sin ningún tipo de concurrencia (un único usuario virtual) y con toda la infraestructura dedicada al mismo. Se ejecutan sobre la infraestructura final en donde se ejecutará el resto de las pruebas e idealmente con la infraestructura dedicada a dichas pruebas para que los resultados no se vean afectados por actividades adicionales. El objetivo, como se mencionó anteriormente, es conocer los tiempos de respuesta de cada transacción en un entorno en donde no se compite con ninguna otra ejecución por los recursos del sistema. De esta manera se obtienen "los mejores" tiempos de respuesta posibles que puede obtener cada transacción con la configuración inicial y de esta manera tener un punto de referencia para analizar el impacto de los distintos escenarios de concurrencia sobre cada transacción es decir al estudiar la evolución de las transacciones cuando la carga vaya en aumento. La diferencia entre los tiempos base y los tiempos obtenidos en las diferentes pruebas brindará un indicativo de la sensibilidad de la misma a la concurrencia. Debido a que estos tiempos son de fundamental importancia para futuros análisis, es necesario ejecutar cada transacción una cantidad suficiente de veces como para obtener datos estadísticamente válidos.

Como se mencionaba anteriormente, la ejecución de los escenarios definidos se realiza de manera gradual. Se debe definir cómo se incrementará la carga de acuerdo a la realidad del proyecto, aunque generalmente se aplica un incremento del 20% en cada prueba. Por este motivo se ejecuta el 20% de la carga, luego el 40%, el 60%, el 80% hasta llegar al 100% que es el escenario objetivo. Esta estrategia de ejecución que propone el proceso trae aparejado varias ventajas. La primera es que los problemas de desempeño más evidentes del sistema aparecen en porcentajes de carga bajos en donde la complejidad de la prueba es menor. Luego a medida que se incrementa la carga se van atacando problemas cada vez más sutiles hasta llegar a cumplir los objetivos planteados. Por otro lado, en las primeras ejecuciones, de baja carga, se van ajustando algunos detalles de infraestructura tales como las herramientas de monitorización, los datos de prueba o incluso los propios scripts de prueba. También el personal involucrado en las pruebas va adquiriendo la metodología de ejecución de manera paulatina, lo cual es preferible antes que comenzar con la complejidad que puede implicar el escenario objetivo.

La Figura 4 muestra la dinámica que se sigue en esta etapa para cada una de las ejecuciones.

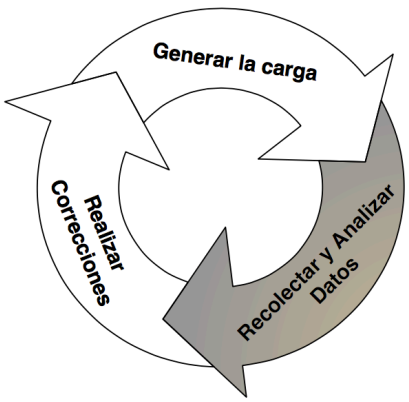



**Figura 4: Ciclo de ejecuciones**

Denominamos “Generar la carga” a lo que implica preparar la ejecución de un escenario de prueba (configuración y datos) y la ejecución propiamente dicha. La prueba puede presentar fallas u otro problema y sea más productivo frenarla. Una vez que finaliza la prueba se recolectan datos que permitan analizar los resultados de la ejecución. Generalmente estos son *logs* de monitorización que permiten generar gráficas con la evolución de los contadores a lo largo del tiempo y los *logs* de la generación propiamente dicha que permiten estudiar la cantidad y tipo de errores que se produjeron y generar gráficas de la evolución de los tiempos de respuesta y carga, en lo que respecta a cantidad de usuarios concurrente, que tuvo el sistema a lo largo del tiempo de la prueba. Con estos datos y la experiencia del *tester*, mejor aún si cuenta con un equipo multidisciplinario con expertos en las tecnologías del sistema bajo pruebas, se puede reportar el análisis. Este reporte no tiene por que ser un documento formal, el objetivo es la presentación de los resultados. En muchos casos alcanza con un correo electrónico o mostrar distintos programas correlacionando datos (ej. tiempos de respuesta vs. uso de CPU de determinado equipo).

Luego de que se ejecutó cada escenario con todos los monitores definidos activados, se debe recolectar y analizar la información. Es importante prestar atención a los distintos datos, tanto del lado de las generadoras de carga como del resto de la infraestructura. El dato más insignificante puede ser la causa de un problema. Por este motivo es de vital importancia para el éxito del proyecto contar con expertos en cada uno de los componentes con el fin de poder rápidamente focalizarse en la información relevante. Es necesario en este tipo de pruebas poder interrelacionar los distintos indicadores para analizar globalmente el sistema y localizar cual es la pieza que está convirtiéndose en el cuello de botella. Hay que tener en cuenta que los tiempos de respuesta son sólo los síntomas, el desafío es encontrar las causas de esos síntomas.

Más allá de los indicadores recopilados, el proceso sugiere que un experto funcional de la aplicación utilice el sistema durante la prueba. Esto tiene como objetivo obtener una retroalimentación subjetiva del comportamiento del sistema, ya que muchas veces cuesta visualizar si el tiempo en segundos es adecuado o no para la operativa del negocio. A esto se le da en el proceso el nombre de monitor humano.

Luego que se ha analizado toda la información resultante de las pruebas y confirmado los datos obtenidos, se debe comunicar al resto de las partes involucradas. Con este fin se puede preparar un reporte formal con el detalle de toda la información y/o pequeños informes o gráficas que varíen de acuerdo al público que recibirá dicho material. Hay que recordar que el personal involucrado en una prueba de desempeño tiene conocimientos heterogéneos y espera recibir información útil de la prueba de acuerdo a sus intereses.

Finalmente, se realizan correcciones y se ajusta el sistema para una próxima corrida de pruebas. Las mejoras al desempeño del sistema se pueden lograr principalmente por dos vías: mejoras en la lógica o ajustes a la infraestructura. El análisis de los datos tiene que haber arrojado información suficiente para saber por dónde abordar la corrección o mejora.

## 2.7. Etapa final

En esta etapa se le da un cierre al proyecto de pruebas de desempeño tanto para el equipo de pruebas como para el interesado en los resultados (ej. cliente que contrató el proyecto). Con este fin se sugiere realizar un informe final del proyecto, en el cual se detalla la metodología seguida a lo largo del mismo, las ejecuciones realizadas, las mejoras efectuadas sobre el sistema y una serie de recomendaciones para seguir mejorando el desempeño del sistema. Otro punto importante de esta etapa es realizar una recopilación de toda la información y artefactos de prueba creados a lo largo del proyecto con el fin de respaldarlo para su eventual uso en el futuro o como material de consulta. Es recomendable realizar una presentación para una transferencia tecnológica de las herramientas utilizadas y de la metodología seguida al resto del equipo y a todo aquel interesado en aprender de los problemas encontrados y soluciones aplicadas. Esta actividad tiene el objetivo

de proveer el conocimiento necesario para que el interesado pueda seguir con la ejecución de las pruebas en caso de así requerirlo.

Posteriormente, el equipo de pruebas realiza el denominado análisis post-mortem. El mismo consiste en una evaluación general del proyecto para analizar las desviaciones con respecto a la estimación, el cumplimiento de los objetivos del proyecto y las oportunidades de mejora de la metodología seguida.

En los siguientes capítulos se presenta tres casos de estudio donde se aplica la metodología los cuales abordan tres tipos de problemas vinculantes a sistemas críticos en diferentes contextos. Los sistemas bajo pruebas son implementados con distintas tecnologías que implican superar desafíos tanto desde el punto de generación de la simulación como de solucionar los problemas que se detectan. Dichos casos de estudio fueron seleccionados de entre más de 20 proyectos por su riqueza tecnológica y metodológica, uso de herramientas e impacto en la ciencia, tecnología y sociedad. Fueron realizados en el marco de investigación y desarrollo de problemas reales llevados adelante por el autor en el seno del CES. La componente de investigación y la metodología aplicada fueron clave para el resultado exitoso de cada uno de ellos. En forma sucinta los problemas seleccionados fueron:

- En el Capitulo 3 “Fusión de dos bancos”.
- En el Capitulo 4 “Central telefónica”.
- En el Capitulo 5 “Aplicación en la salud”.

# 3. Fusión de dos bancos

El primer caso es sobre el sistema Bantotal utilizado en producción para el procesamiento de las transacciones financieras por varios bancos, particularmente este caso es para la fusión de Caja Nuestra Gente (CNG) y Financiera Confianza (FC) del Perú. Dicho proyecto se realizó desde Julio hasta Octubre de 2012, tanto en las instalaciones de CNG en Lima como en las oficinas de CES en Montevideo.

La prueba de performance se basó en un estudio del escenario estimado de uso de la aplicación, por lo tanto para la carga esperada en el sistema se utilizó como referencia la operativa que se llevaba adelante en ambas instituciones. Como consecuencia de esto, se analizaron más de 10 transacciones, algunas de las cuales incluían varios ciclos funcionales, motivo por el cual se separaron.

Este capitulo consta de una descripción detallada de las pruebas realizadas, incluyendo los escenarios definidos, el ambiente de prueba y las métricas a recopilar. La primera sección, describe las pruebas desde el punto de vista de sus objetivos, tanto en cuanto al software que se probó como al objetivo específico de la prueba (ver "Objetivos"). Se repasa y verifica, a grandes rasgos, que la metodología utilizada para llevar a cabo el proyecto es la presentada en el capitulo anterior. La sección "Escenarios" describe todas las características de las pruebas desde el punto de vista del entorno de las mismas. Ítems como infraestructura utilizada y escenarios de carga son vistos en dicho apartado. La sección "Automatización y armado del ambiente de pruebas" describe aquellos temas relativos a la robotización de la prueba, punto neurálgico en la realización de una prueba de performance. En "Monitorización" se describen las herramientas utilizadas para realizar dicha actividad. "Ejecución de pruebas" describe las actividades en donde se ejecutaron los scripts antes definidos y se monitoreo la infraestructura. Finalmente, en "Conclusiones" se realizará un recuento de los resultados obtenidos y los pasos a seguir en el futuro para lograr mejoras en la performance del sistema Bantotal.

## 3.1. Descripción de las pruebas

En esta sección, se describe el objetivo del ensayo y el software sobre el que se aplicó el mismo. Estos puntos pondrán en contexto a estas pruebas y explicarán muchas de las decisiones tomadas durante la planificación del proyecto; resumen la "Etapa inicial" que surge de mantener reuniones preliminares, implementar la prueba de concepto y presentar una propuesta de trabajo.

### *3.1.1. Sistema Bajo Pruebas*

El SUT es Bantotal, desarrollado por De Larrobla y Asociados en su versión web. Se trata de un sistema de procesamiento de transacciones financieras adaptado para la fusión de Caja Nuestra Gente y Financiera Confianza, desarrollado en Genexus 8.0 generando código Java. El sistema consiste en una arquitectura orientada a servicios (tres capas).

| | |
|---|---|
| **Software** | Bantotal web |
| **Arquitectura** | 3 capas |
| **DBMS** | Oracle 11.2.0.2 |
| **Lenguaje de desarrollo** | Genexus 8.0 generando Java |

### 3.1.2. Contexto

El sistema será utilizado de manera distribuida en todas las oficinas. La presente prueba de performance tuvo lugar previo a la implantación del mismo y pretendió simular el uso del sistema por parte de los usuarios en las diferentes sucursales.

### 3.1.3. Objetivos

Esta prueba se desarrolló con el objetivo de verificar la performance del sistema, estudiar el comportamiento del mismo ante cargas de uso para las cuales el sistema fue diseñado y encontrar el punto de quiebre así como también detectar y solucionar problemas de performance y validar una nueva plataforma de ejecución. En pocas palabras, verificar que los requisitos no funcionales de performance del sistema puedan cumplirse y particularmente que los tiempos de respuesta de las transacciones sean satisfactorios para la operativa diaria y que el sistema se mantenga estable. Además, se busca determinar si todos los componentes de la infraestructura se comportan de manera adecuada a lo largo de la ejecución.

Este estudio implica una serie de actividades y de verificaciones a realizarse, entre las cuales se pueden enumerar:

1. Tiempos de respuesta de los programas del sistema. Estos tiempos pueden ser agrupados en transacciones o sectores del ciclo funcional.
2. Desempeño del servidor de aplicación ante distintas situaciones de carga.
3. Desempeño del servidor de bases de datos ante distintas situaciones de carga.

La verificación de estos parámetros cumple la finalidad de detectar y solucionar problemas de performance y determinar el uso de la infraestructura de manera que dichos datos puedan ser utilizados en el *sizing* de futuras implantaciones del sistema.

Se estudiará el sistema con el objetivo de verificar el correcto funcionamiento en lo que refiere a performance de la nueva plataforma implantada, previo a la fusión que se llevará a cabo.

Los requisitos no funcionales del sistema fueron establecidos en función de la cantidad de operaciones y usuarios nominales del sistema en un lapso, estableciendo como la ventana de tiempo de las pruebas “una hora”. Se simularon los momentos pico que se mencionaron anteriormente. Estas definiciones pueden verse en la sección “Escenario de carga”.

## 3.2. Escenarios

Se describe a continuación los escenarios bajo los cuales se desarrollan las pruebas que surgen del “Análisis de requerimientos”. Estos escenarios se componen tanto del entorno de infraestructura que se utiliza como de los distintos escenarios de carga a reproducir con la herramienta de generación de carga.

Estas combinaciones marcan la validez de las pruebas y de los resultados a obtener en las mismas.

### 3.2.1. Escenario de infraestructura

Para la realización de las pruebas se utilizó la infraestructura que se encuentra esquematizada en la *Figura 5,* la cual será la utilizada por el sistema una vez se encuentre en producción. Se puede notar la existencia de diferentes roles en los equipos utilizados:

- **Generadores de carga (OpenSTA):** son los responsables de generar la carga (pedidos HTTP) de cada uno de los usuarios virtuales que participan en la prueba.

- **Balanceadoras de carga:** encargadas de gestionar la carga dirigida hacia los servidores de aplicación, balanceando las peticiones recibidas al servidor que corresponda.
- **Servidores web:** si bien estos servidores existen en la infraestructura, actúan como "pasamanos" ya que no fueron configurados para servir información.
- **Servidor de aplicación (WAS):** Es el responsable de realizar el procesamiento de los pedidos HTTP por parte de los usuarios virtuales que se incluyen en la prueba.
- **Servidores de base de datos (Oracle):** Es quien oficia de host para el DBMS que contiene los datos del sistema, en este caso Oracle 11.2.0.2. Están conectados en una red con el servidor de aplicación.
- **Storage externo:** Es el encargado del almacenamiento de los datos.

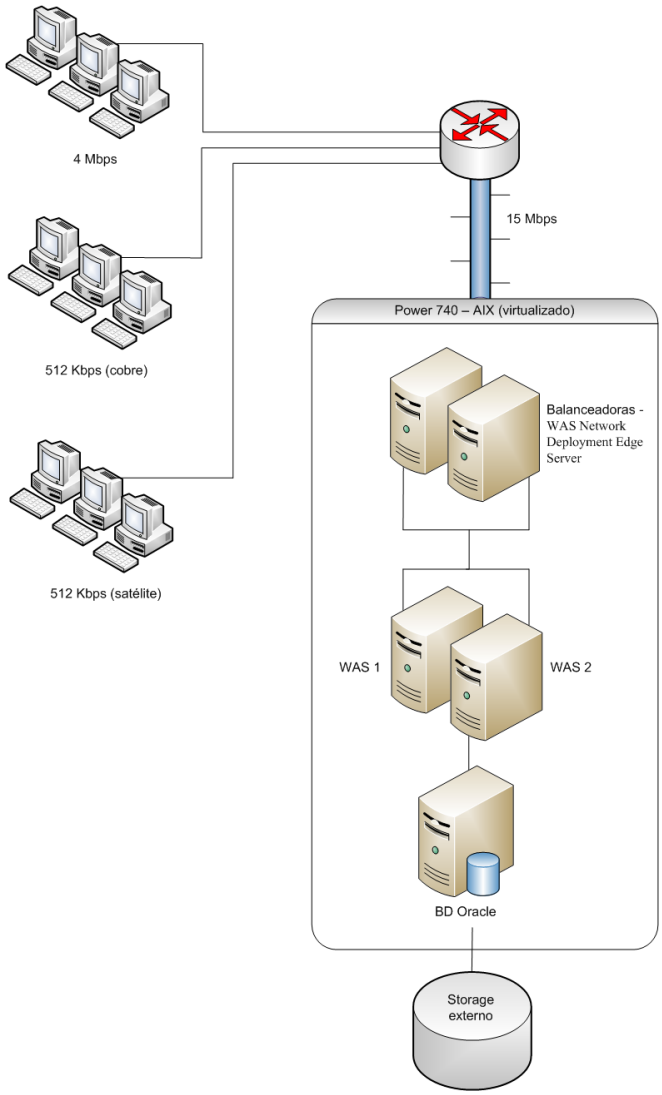



**Figura 5 - Infraestructura**

Todos los componentes del SUT ejecutan sobre un servidor Power 740 con AIX virtualizado. A su vez, cada uno de los componentes virtualizados también ejecutan sobre plataformas AIX.

Para las pruebas se desplegó el sistema en una maqueta similar, y que será utilizada en producción, donde las computadoras que utilizan los usuarios fueron sustituidas por varios PC configurados con una herramienta generadora de carga, OpenSTA, que emula la carga que los

usuarios generan. En la siguiente figura se muestra un esquema de infraestructura similar al presentado anteriormente en la cual se observa el esquema de simulación, en el mismo se eliminan los usuarios (clientes) e incluyen las generadoras de carga como remplazo para simular la cantidad de usuarios definida. La presencia de "usuarios testigos" reales accediendo a la aplicación a través del navegador, en una sub-red aislada de la que incluye la generadora de carga confirma subjetivamente lo que la herramienta registra de forma objetiva. Un "usuario testigo" automatizado, lo hace de forma subjetiva.

El ambiente donde se ejecutaron las pruebas, "Modulares", estuvo salvo cuando se indica lo contrario dedicado, es decir, sin competencia por los recursos con otras actividades ajenas a las pruebas de performance.

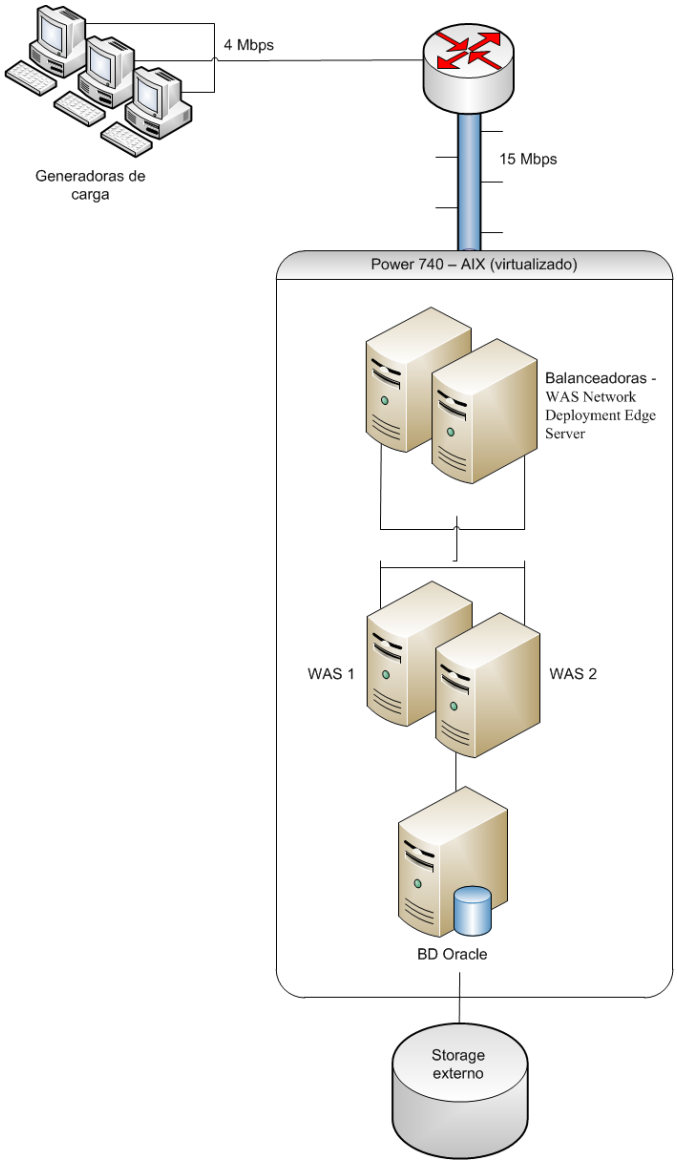



**Figura 6 – Infraestructura de simulación**

## Hardware y Software Base

En esta sección se describen características de hardware y software de los equipos.

**Generadoras de carga**

Inicialmente se utilizaron varios equipos, tipo PC, para la simulación de la carga. Las características de cada equipo se pueden ver en las siguientes tablas:

| Hardware |
|---|
| CPU Dual Core 2.8Hz |
| 2GB RAM |
| Disco SATA 300GB 5400 RPM |

| Software |
|---|
| Windows XP Professional SP3 |
| OpenSTA 1.4.04 |

Estas generadoras de carga inicialmente se conectaron al sistema utilizando una LAN de 4Mbps. En las primeras pruebas se detectó que la red era un cuello de botella para la carga que se emulaba. Por este motivo, luego se decidió generar la carga directamente desde el centro de cómputos, haciendo que las generadoras de carga pasaran a ser máquinas virtuales alojadas en un mismo servidor físico ubicado en el centro de cómputos. De esta manera se conectaban al sistema utilizando una LAN de 10Gbps. Las características de las mismas eran las siguientes:

| Hardware |
|---|
| CPU Dual Core 2.8Hz |
| 2GB RAM |
| 30GB de disco |

| Software |
|---|
| Windows Server 2003 SP3 |
| OpenSTA 1.4.04 |

**Servidor host (virtualización)**

| Hardware |
|---|
| Power 740 16 Cores de 3.55 GHz |
| 96GB RAM |
| Disco SATA 300GB 5400 RPM |

| Software |
|---|
| AIX 6100-05-06-1119 (version 6.1, tecnolgía 5) |
| HMC V7 R7.4.0.0 (consola Manager del Power) |

Los siguientes componentes, como se comentó, son virtuales y ejecutan sobre el hardware detallado anteriormente, por lo que se describirá únicamente la cantidad de recursos hardware asignados y los detalles de los elementos software presentes en cada uno.

**Balanceadoras de carga (x2)**

| Recursos asignados |
|---|
| 0.5 cores |
| 2GB RAM |

**Servidores de aplicación (x2)**

| Recursos asignados |
|---|
| 3.5 cores |
| 4GB RAM |

| Software |
|---|
| WebSphereApplication Server 7.0.0.21 |

**Servidor de base de datos**

| Recursos asignados |
|---|
| 6 cores |
| 64GB RAM |

| Software |
|---|
| Oracle Enterprise Edition 11.2.0.2 |

### *3.2.2. Escenario de carga*

Sobre el escenario de infraestructura establecido, se definió el escenario del 100% de la carga esperada para la operativa diaria en la hora pico que se detallan en esta sección.

El escenario está compuestos por:

- Transacciones que participan en la prueba
- Usuarios del sistema.
- Cantidad de usuarios que ejecutan cada una de las transacciones.
- Cantidad de transacciones que ejecutan en una hora (*Throughput*) para cada transacción.
- Tiempos en los que los usuarios del sistema no interaccionan (*Think times* y *Delays*)

El escenario fue definido en conjunto con CNG, basado en estadísticas del uso del sistema actual y del uso previsto del sistema a implantar. Para su definición, se tuvieron en cuenta:

- Cantidad de ejecuciones de las transacciones.
- Cantidad de usuarios concurrentes.
- Importancia de cada transacción en la operativa.
- Consumo de recursos estimado de la transacción.

#### 3.2.2.1. Transacciones seleccionadas

En un principio se seleccionaron ocho transacciones a incluir en las pruebas. Luego se agregaron cinco transacciones más que resultaron importantes para la emulación de la operativa diaria de CNG en cuanto a su frecuencia de uso. Esto implicó más trabajo del planificado y se priorizó su automatización. El conjunto completo de potenciales transacciones a incluir en la prueba se pueden observar en la siguiente tabla, en gris aquellas que fueron consideradas de baja prioridad para incluirlas en las primeras pruebas debido a su madurez en lo que respecta a desarrollo.

| Transacción | #Pasos (interacción SUT con el usuario) |
|---|---|
| TRN01_RETIRO_EN_CUENTAS_PASIVAS | 8 |
| TRN02_DEPOSITO_EN_CUENTAS_PASIVAS | 7 |
| TRN03_ALTA_DPF | 6 |
| TRN04_CONFIRMACION_DE_ALTA_DPF | 5 |
| TRN05_CANCELACION_DPF | N/A |
| TRN06_SOLICITUD | 24 |
| TRN07_EVALUACION | 51 |
| TRN07_2_CONFIRMACION_EVALUACION | 7 |
| TRN08_APROBACION | N/A |
| TRN09_DESEMBOLSO_DE_CREDITOS | N/A |
| TRN10_AMORTIZACION_(CANCELACION)_DE_DREDITOS | 10 |
| TRN11_COMPRA_DE_MONEDA_EXTRANJERA | 8 |
| TRN12_VENTA_DE_MONEDA_ENTRANJERA | 8 |

En el contexto de las pruebas de performance y con el fin de tener resultados lo antes posible, se da que las pruebas usualmente se inician a medida que se cuenta con un sub-conjunto de transacciones automatizadas correctamente.

En la medida que la plataforma se desempeña de acuerdo a lo esperado, nuestra experiencia en este tipo de pruebas y con esta tecnología nos permite afirmar que, bajo ciertos supuestos, el período y esfuerzo previstos en el cronograma es suficiente para terminar las pruebas integrando las transacciones detalladas. No obstante, si en el devenir del proyecto se detectan problemas complejos en las fases tempranas, es probable que sea mejor concentrar los esfuerzos en resolver esos problemas, aún a expensas de cubrir todo el universo de transacciones.

Es importante definir la importancia, de acuerdo a criterios como ejecución intensiva por parte de los usuarios, consumo intensivo de recursos por parte de la transacción, sospechas de problemas de performance, entre otros, y madurez de cada transacción para no sincronizar la obtención de resultados a la finalización de la automatización y así obtener resultados conjuntamente y muy cerca de la puesta en producción del sistema sino antes. De esta manera se puede prever la paralelización de las etapas automatización, ejecución y reportes.

En este proyecto se fueron integrando las transacciones de acuerdo a la disponibilidad y prioridad antes mencionadas. Finalmente sólo la 5, 8 y 9 se analizaron pero no se automatizaron el resto se fueron integrando en las pruebas. Las razones de esto fueron principalmente que por momentos no estuvo el ambiente disponible y además se realizaban cambios en la programación del SUT que afectaron la correctitud de los scripts generados, esto hacía necesario re-automatizar para corregir los problemas que hacían que fallen, afectando así la planificación.

El detalle de las transacciones automatizadas se encuentra en el “Anexo III – Guiones fusión”.

#### 3.2.2.2. Mezcla de transacciones

Una vez definidas las transacciones, se definió la cantidad de usuarios que ejecutan las mismas basándose en estadísticas de uso del sistema actual y en características de la operativa esperada con el nuevo sistema.

El escenario a simular será el comprendido entre la hora 9am y 10am de la operativa diaria. Esta hora representa la hora pico del mismo en la cual se da el mayor estrés del sistema, por lo que dicho escenario es el considerado más crítico. Para el escenario definido, considerando la mezcla de transacciones, cantidad de usuarios concurrentes y frecuencia de uso, se define el escenario del 100% de la carga, y en base a éste se definen distintas escalas proporcionales, como por ejemplo Escenario del 10%, del 25%, del 50%, del 200%, etc.

El escenario que surge al asignar a cada transacción la cantidad de usuarios virtuales e iteraciones que los mismos ejecutan en una hora se conocerá como "Escenario del 100%". Para lograr que en una hora ejecuten la cantidad de transacciones deseada, se define para cada transacción el tiempo de espera (*delays*) entre cada una de las iteraciones de la operación para cada usuario virtual. Estas esperas se ajustan con los tiempos obtenidos en las ejecuciones de las pruebas previas (generalmente de porcentajes menores de carga). El objetivo es distribuir la cantidad de ejecuciones de los usuarios uniformemente a lo largo de la prueba. Para el escenario de menor carga (generalmente 10%) se parte de los tiempos base obtenidos. El "Escenario del 100%" se puede observar en la siguiente tabla, que detalla la cantidad de usuarios esperados en el sistema para cada una de las transacciones, las iteraciones que cada usuario realizará y el total de datos que se procesarán (el total de transacciones). Los *delays* se conocerán luego de ejecutar los baselines.

| Transacción | Usuarios concurrentes | Iteraciones (Por hora Por usuario) | Total de datos a procesar |
|---|---|---|---|
| TRN01_RETIRO_EN_CUENTAS_PASIVAS | 87 | 32 | 2784 |
| TRN02_DEPOSITO_EN_CUENTAS_PASIVAS | 67 | 32 | 2144 |
| TRN03_ALTA_DPF | 140 | 10 | 1400 |
| TRN04_CONFIRMACION_DE_ALTA_DPF | 27 | 32 | 864 |
| TRN05_CANCELACION_DPF | 27 | 32 | 864 |
| TRN06_SOLICITUD | 650 | 1 | 650 |
| TRN07_EVALUACION | 650 | 1 | 650 |
| TRN07_2_CONFIRMACION_EVALUACION | 1 | 110 | 110 |
| TRN08_APROBACION | 260 | 2 | 520 |
| TRN09_DESEMBOLSO_DE_CREDITOS | 41 | 32 | 1312 |
| TRN10_AMORTIZACION_(CANCELACION)_DE_CREDITO | 148 | 32 | 4736 |
| TRN11_COMPRA_DE_MONEDA_EXTRANJERA | 27 | 32 | 864 |
| TRN12_VENTA_DE_MONEDA_ENTRANJERA | 27 | 32 | 864 |
| **Total** | **2152** | | |

Los escenarios de menor carga (ej: 10%, 25% y 50%) son ejecutados previos a las pruebas con el 100% de la carga esperada y tienen como objetivo estudiar el desempeño del sistema ante distintas cargas y tomar un enfoque incremental que permitiera extraer conclusiones de manera más controlada, a la vez de ir avanzando en las pruebas y haciendo "sintonía" de la aplicación en base a los resultados obtenidos en cada ejecución.

A continuación se muestran los pasos a seguir sobre la aplicación para poder ejecutar la transacción interactiva “TRN03 - Alta de DPF (Depósito a Plazo Fijo)”, en el “Anexo III – Guiones fusión” se pueden ver para el resto de las transacciones automatizadas. Queda bien definido el flujo a ejecutar y de esta forma bien determinadas las acciones que ejecutarán los robots. En las siguientes tablas se observan las acciones a realizar y el resultado esperado de esta acción, indicándose además el *think time* definido para cada paso.

TRN03 - Alta de DPF (Depósito a Plazo Fijo)

| Realizar acción | Resultado | Think time |
|---|---|---|
| Login (Promotor de Negocios) | | Bajo |
| Seleccionar el menú "Inicio -> Menú de DPF -> Apertura de DPF" | Va a la pantalla de "Apertura Depósito a Plazo" | Bajo |
| Ingresar (Cuenta cliente, Moneda = 0, monto = 500, Apert. DPF Afecto a ITF = "Si afecto. Incluido en Capital", Tipo de DPF = "Interés al Vencimiento", Plazo = 30, F/ Vencimiento = NADA, Tasa no tocar pero cuidado porque se genera (al presionar validar), Forma de Pago = Efectivo, Moneda = 0, Instrucciones = "Cancelar y Acreditar al vto.", Tipos de Acreditación = "21: caja de ahorros", Ejecutar = check, comentario = "CES_COMENTARIO" y presionar validar) | Calcula la tasa | Alto |
| Presionar "Confirmar" | Pone en rojo la palabra CONFIRMACION | Bajo |
| Presionar "Confirmar" | Pone el texto: "Movimiento a ser confirmado; relación: 2" | Bajo |
| Presionar "Continuar" | Vuelve a la pantalla: "Apertura Depósito a Plazo" | Bajo |

### *3.2.3. Datos necesarios*

Las pruebas se hicieron con bases de datos con un volumen de equivalente al día en que el sistema salía en producción

## 3.3. Automatización y armado del ambiente de pruebas

### *3.3.1. Herramientas*

La herramienta seleccionada para automatizar las transacciones que utilizan el protocolo HTTP ha sido OpenSTA. Al tratase de una herramienta libre que se encuentra bajo licenciamiento GPL su utilización no tuvo costo adicional para el proyecto.

La utilización de esta herramienta ha sido posible debido a que el tráfico entre el cliente web y el servidor de aplicaciones es siempre HTTP.

Para visualizar en qué punto se incorpora la herramienta de generación de carga a la infraestructura del sistema referirse a la *Figura 6* correspondiente a la infraestructura de simulación.

### 3.3.2. Reproducción del escenario

Los escenarios se reproducen utilizando las capacidades que OpenSTA provee, oficiando de orquestador para los distintos programas de manera de simular correctamente todos los usuarios que hacen uso del sistema.

Cada uno de los scripts con OpenSTA se desarrolló de la siguiente manera:

1. Grabación original del scripts.
2. Validación de cada uno de los pasos del script. Esto implica la validación de cada una de las pantallas que tiene el sistema.
3. Generalización de los scripts. Esto implica reemplazar constantes que se tienen en el momento de la grabación (usuarios, cuentas, entre otros) por variables a ser tomadas desde un archivo por cada una de las ejecuciones del script.
4. Definición de la sección de login. Con esto se logró que cada una de las ejecuciones de los scripts por parte de los usuarios virtuales no realizaran el login. Esto imita lo que un usuario real realiza. Cada usuario realiza el login una única vez, manteniendo por lo tanto una única sesión en el sistema. Sin embrago, ante la falla de la transacción (la cual se da por datos erróneos o errores que se dieran en la aplicación), el usuario vuelve a realizar el login.

Se definieron, en primera instancia, la misma cantidad de usuarios reales como usuarios virtuales (ver "Escenario de carga"). Cada uno de ellos ejecuta una cantidad determinada de veces cierta transacción. Esto es una simplificación de la realidad ("usuarios virtuales especializados") que a los efectos de estudiar la performance es despreciable. La cantidad de transacciones que cada uno de ellos ejecutó fue fijada a partir de datos definidos con los distintos actores que conocen del negocio. Como se comentó anteriormente entre cada una de las ejecuciones se define un tiempo de espera que tiene como propósito ajustar la cantidad de transacciones que cada usuario realiza en el plazo de la prueba. Los usuarios ingresan a la prueba con pequeños retrasos de manera que no se generen sincronizaciones en los pedidos, con el objetivo de simular de mejor forma la realidad (período de *ramp-up*).

Para emular el ingreso de usuarios al sistema y estudiar la carga, se definió en primera instancia que el total de los usuarios ingresen al sistema en 15 minutos. Todas las pruebas realizadas cuentan con monitorización del sistema a fin de observar el comportamiento de los componentes involucrados y medir su rendimiento. En los casos en que se detectan patrones a analizar, se aumentan los niveles de *log* y monitores. La monitorización se describe a continuación.

### 3.3.3. Monitorización

Durante el despliegue del "Escenario de infraestructura" definido para la simulación (es decir el "Armado del ambiente de pruebas") también se configuraron los monitores de primer nivel que recolectaron datos para detectar posibles cuellos de botella.

#### 3.3.3.1. Indicadores

Los siguientes indicadores fueron recolectados y estudiados luego de cada prueba. En este documento se presentan algunos elementos del monitoreo que facilita la comprensión de cada situación.

- **Servidores - Sistema Operativo (sistemas *hosts* y *guests*):** Uso de CPU, uso de memoria, uso de la red, uso de IO.
- **Manejador de Base de Datos:** Top de SQLs, tiempos de ejecución de SQL, Locks.
- **Servidor de aplicación – WebSphere Application Server (WAS):** Uso de memoria (*heap* de la JVM), Tiempos de *Garbage Collection*.
- **Comunicación con el centro de cómputos:** Ancho de banda utilizado.

#### 3.3.3.2. Herramientas

Se utilizan varias herramientas para realizar la tarea de monitorización:

- **nmon** para los recursos del el SUT: sistemas *host* (Power 740) y *guest* (virtuales).
- **JConsole** para el uso de CPU y memoria (*heap* y *garbage collections*) de la JVM del servidor de aplicación, así como también para analizar la actividad con la base de datos.
- **NetFlow Analyzer** y **RRDTool** para ancho de banda.
- **Perfmon** para los recursos de las generadoras de carga.

## 3.4. Ejecución de pruebas

En esta sección se describen las pruebas realizadas desde el día domingo cinco de agosto hasta el día martes nueve de octubre y los resultados obtenidos de las mismas. Las pruebas fueron realizadas desde Uruguay accediendo a las generadoras de carga en CNG mediante escritorio remoto. La infraestructura utilizada fue la misma que se utiliza en producción.

### *3.4.1. Cronograma de las pruebas*

El cronograma suponía tres semanas de ejecuciones a partir de la tercer semana del proyecto.

| | **Semana** | | |
|---|---|---|---|
| **Etapa** | **3** | **4** | **5** |
| Baselines 2 días - (7 al 8 de agosto) | ■ | | |
| 10% 3 días - (8 al 11 de agosto) | ■ | | |
| 20% 1 día - (14 de agosto) | | ■ | |
| 50% 1 día - (15 de agosto) | | ■ | |
| 100% 2 días - (16 al 18 de agosto) | | ■ | |
| +100% 5 días - (21 al 25 de agosto) | | | ■ |

Las pruebas se realizaron a lo largo de ocho semanas aproximadamente sin embargo el esfuerzo en horas del equipo de pruebas fue el equivalente a las tres inicialmente planificadas.

El lapso estimado para la ejecución de algunas pruebas generalmente es más que el planificado, por ejemplo en este caso la primer prueba en concurrencia (10%). Esto puede deberse por el inadecuado uso del ambiente y porque a priori no se conoce cuántos problemas pueden evidenciarse en las pruebas así como tampoco el tiempo que llevará su detección y solución.

| | Semana | | | | | | |
|---|---|---|---|---|---|---|---|
| **Etapa** | **3** | **4** | **5** | **5** | **6** | **7** | **8** |
| | | | | | | | |
| Baselines 2 días - (5 al 7 de agosto) | ■ | | | | | | |
| 10% 5 días - (8 al 22 de agosto) | | ■ | ■ | | | | |
| 50% 1 día - (30 de agosto) | | | | ■ | | | |
| 100% 2 días - (30 de agosto al 19 de setiembre) | | | | ■ | | ■ | |
| 200% 2 días - (18 al 20 de setiembre) | | | | | | ■ | |
| 150% 8 días - (20 de setiembre al 8 de octubre) | | | | | | ■ | ■ |
| | | | | | | | |

Durante las pruebas se detectaron varios problemas, entre ellos, durante la ejecución:

- quedaban, al menos 10, requerimientos lanzando un número gigantesco de sentencias por segundo. Al estudiar el problema se detectó que un programa entraba en *loop*. Esto hacía que los tiempos de respuesta aumentaran.
- el sistema tarda demasiado en responder y arroja el error "Internal Server Error". Al estudiar el problema se detectó que era para algunos datos.

- el balanceo era ineficaz, dado que un servidor web prácticamente no recibía carga y el otro estaba sobre-cargado.

Los cambios generalmente implicaban mejoras en los tiempos de respuesta, sin embargo estos seguían altos y la carga sobre el SUT era mínima. Se analizó el canal de comunicación con el centro de cómputos, se detectó que se estaba consumiendo prácticamente completo por la carga generada. Se movieron físicamente las generadoras dentro del centro de cómputos para que puedan generar la carga a velocidad LAN. Esto era el cuello de botella y permitió avanzar con la carga.

A titulo ilustrativo, para este primer caso, se presenta la "bitácora" (ver *"*Anexo IV – Bitácora*"*) conteniendo un resumen ordenado cronológicamente de los resultados más importantes de algunas ejecuciones de pruebas. Se realizaron más pruebas que las enumeradas en esa sección, las mismas tuvieron un alto porcentaje de fallas, tenían como objetivo probar algún cambio de monitorización o de automatización, faltó monitorizar algún componente al momento de realizar la prueba o son repeticiones de pruebas para verificar comportamientos y no contaron con aportes significativos.

### *3.4.2. Resultados*

A continuación, se mostrarán los resultados más relevantes de las pruebas realizadas al sistema Bantotal. No todas las ejecuciones son descriptas en esta sección, sino aquellas que arrojaron resultados válidos o que mostraron el comportamiento ante un cambio sobre la infraestructura o sobre la lógica de la aplicación.

Para cada transacción se muestra una tabla que incluye los tiempos de respuesta obtenidos para cada uno de los pasos que conforman la transacción. Generalmente se presentan tanto promedios como percentiles 90. El percentil 90 define, en *timers* ordenados de menor a mayor, el valor 90% de la muestra es decir el 10% de los valores para dicho *timer* están por encima del valor del percentil. Un percentil cercano al valor de la media nos muestra una distribución con poca dispersión, lo cuál indica que las distintas ejecuciones de la transacción tienen tiempo aproximados a dicha medida. Además, incluye los tiempos de la transacción global presentados de tres formas diferentes:

1. Transacción completa - Tiempo de procesamiento: tiempo total de la transacción considerando todos los pasos que la conforman, sin incluir tiempos de *thinktime*. Se suele denominar tiempo de procesamiento debido a que establece efectivamente el tiempo de servidor (y eventualmente red), es decir, el tiempo en que el sistema estuvo procesando solicitudes para esa transacción particular.
2. Transacción completa *baselines*: tiempo total de la transacción durante las pruebas *baseline*, por lo que se incluyen los *think times* fijos utilizados para estas pruebas con el objetivo de disminuir el tiempo de ejecución. En definitiva, se corresponde al tiempo de procesamiento adicionándole una cantidad fija correspondiente a la suma de los *think times* fijos en cada paso (este último valor se corresponde con el nombre de referencia "Thinktimes baselines – TOTAL" en las tablas presentadas).
3. Transacción completa carga: tiempo total de la transacción en el mejor caso. Este tiempo incluye los *think times* reales utilizados para la ejecución de los incrementos de carga. En definitiva, se corresponde al tiempo de procesamiento adicionándole una cantidad fija correspondiente a la suma de los *think times* reales en cada paso (este último valor se corresponde con el nombre de referencia "Thinktimes carga – TOTAL" en las tablas presentadas).

#### 3.4.2.1. Baselines

La prueba de *baselines* consistió en la ejecución secuencial de cada una de las transacciones por separado. Cada una de ellas se ejecutó, con un único usuario virtual, aproximadamente 25 veces (iteraciones).

Para agilizar la ejecución de estas pruebas se configuraron *think times* de cinco segundos, más cortos que los utilizados para la ejecución de los incrementos de carga. Al no haber concurrencia no repercute en los resultados, es una constante en el tiempo de respuesta del total de la transacción.

Las ejecuciones de carga se realizaron con los *think times* más realistas a fin de modelar mejor la realidad. Al final de esta sección, ver "Resumen baseline", se realizan cálculos para obtener los tiempos de respuesta de cada transacción con y sin *think times*, los cuales permiten comparar la evolución a lo largo de las distintas pruebas ejecutadas.

Esta prueba también permitió ajustar algunos temas de ambiente y automatización, a modo de ejemplo: algunos de los datos utilizados en la TRN01 y TRN02 no sirven debido a que algunas de las cuentas no tienen subcuentas asociadas. Además, para algunas de las cuentas obtenidas para la TRN02 se saltea el paso de "Trabajar con Gastos", lo cual hace que el *script* falle. Para las posteriores ejecuciones se solicitan cuentas para las cuales se muestre siempre la pantalla de "Trabajar con Gastos".

Siguiendo con el análisis de la transacción "TRN03 - Alta de DPF (Depósito a Plazo Fijo)" a continuación se muestra el análisis que se realizó para cada una de las transacciones a incluir en la prueba (el "Anexo V – Baselines" se puede ver para el resto de las transacciones automatizadas). Se presenta una tabla y una gráfica que resume los resultados. La tabla incluye tiempos de respuesta obtenidos por cada uno de los pasos que conforman la transacción y también los tiempos globales de la transacción. La gráfica muestra la evolución de esos tiempos de respuesta. El color de cada línea está asociado a el paso de la transacción que indica en la leyenda del gráfico. La línea roja presenta la evolución de los tiempos para la transacción completa considerando los *think times* fijos utilizados.

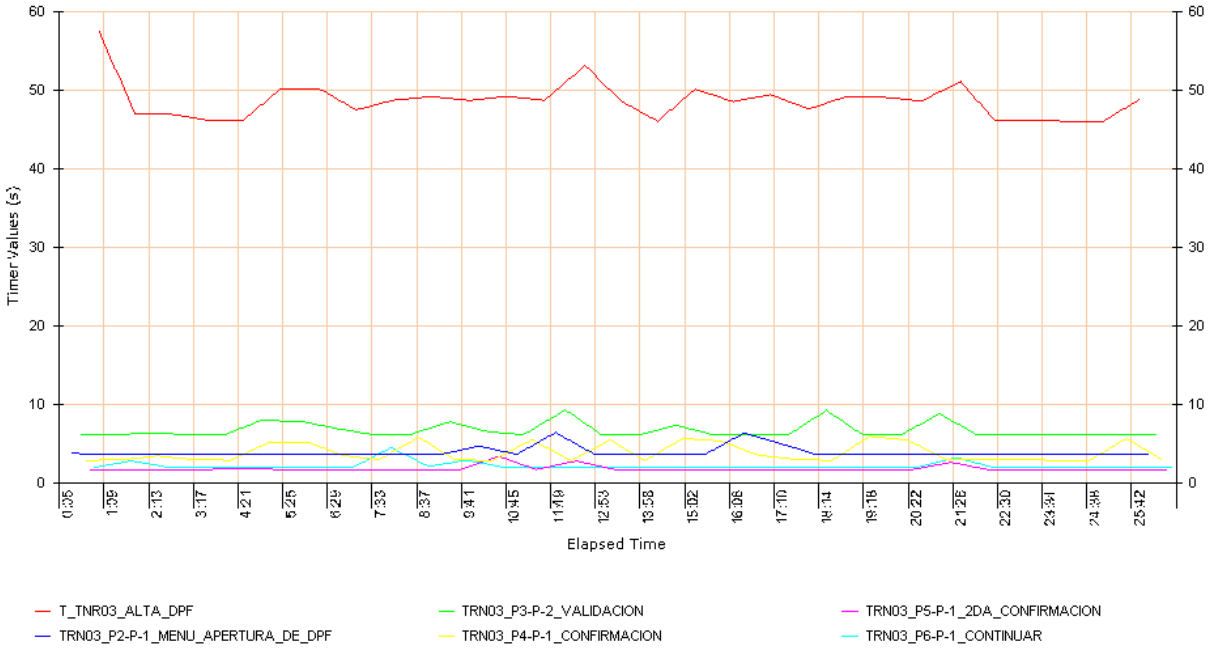
##### TRN03 - Alta de DPF (Depósito a Plazo Fijo)

Los resultados obtenidos son los que se presentan en la siguiente tabla. La misma detalla los tiempos de respuesta promedio obtenidos así como también los percentiles 90, para cada uno de los pasos y para la transacción completa.

| Nombre paso | Timer | *Promedios (segs)* | *Percentiles 90 (segs)* |
|---|---|---|---|
| 1) Loguearse a la aplicación | TRN03_P1-P-1_ACCESO_PDN | 0.38 | 0.00 |
| 2) Seleccionar el menú Apertura de DPF | TRN03_P2-P-1_MENU_APERTURA_DE_DPF | 3.84 | 4.67 |
| 3) Ingresar datos y Validar | TRN03_P3-P-2_VALIDACION | 6.63 | 7.97 |
| 4) Confirmar | TRN03_P4-P-1_CONFIRMACION | 3.78 | 5.60 |
| 5) Confirmar | TRN03_P5-P-1_2DA_CONFIRMACION | 1.78 | 1.79 |
| 6) Continuar | TRN03_P6-P-1_CONTINUAR | 2.11 | 2.76 |
| | Transacción completa - Tiempo de procesamiento (sin thinktimes) | 18.51 | 22.78 |
| | Transacción completa baselines (con thinktimes) | 48.51 | 52.78 |
| | Transacción completa carga (con thinktimes) | 343.51 | 347.78 |

| | |
|---|---|
| Thinktimes Baselines - TOTAL | 30 |
| Thinktimes Carga - TOTAL | 325 |

La *Gráfica 1* traza una recta entre los tiempos de respuesta obtenidos en cada ejecución (muestras de los tiempos obtenidos) para esta transacción. El eje horizontal representa el tiempo de prueba en minutos y en el eje vertical el tiempo de respuesta obtenido en segundos.




**Gráfica 1: Resumen tiempos de respuesta *baseline* TRN03 Alta de DPF**

Como se observa, el tiempo total de la transacción durante los *baselines* fue estable, manteniéndose en aproximadamente 50 segundos considerando los *think times* fijos utilizados para esta ejecución, y considerando los *think times* reales es de aproximadamente 343 segundos. El paso de validación se acerca en algunas muestras a los 10 segundos.

## Resumen baseline

En la siguiente tabla se presentan, a modo de resumen, los tiempos de respuesta obtenidos para la ejecución de los *baselines* por cada una de las transacciones que se incluyeron en las pruebas. La tabla presenta en segundos tanto los tiempos considerando *think times* fijos, reducidos, como los reales que se utilizarán para comparar con los resultados de las ejecuciones de los incrementos.

| Transacciones | Con TT baseline | Con TT reales |
|---|---|---|
| TRN01_RETIRO_EN_CUENTAS_PASIVAS | 60.9 | 85.9 |
| TRN02_DEPOSITO_EN_CUENTAS_PASIVAS | 57.4 | 82.4 |
| TRN03_ALTA_DPF | 48.5 | 343.5 |
| TRN04_CONFIRMACION_DE_ALTA_DPF | 26.8 | 51.8 |
| TRN11_COMPRA_DE_MONEDA_EXTRANJERA | 57.7 | 57.7 |
| TRN12_VENTA_DE_MONEDA_EXTRANJERA | 16 | 70 |

#### 3.4.2.2. 10% de la carga

Una vez que se ejecutaron los *baselines* que se tomaron como referencia para las siguientes ejecuciones, se ejecutó el 10% de la carga esperada. Esto equivale a aproximadamente 180 usuarios haciendo uso de la aplicación en forma concurrente. En un principio se ejecutó el 10% de la carga con las transacciones de la siguiente tabla.

| Transacción | Usuarios concurrentes | Iteraciones (Por Usuario Por hora) | Total de datos a procesar |
|---|---|---|---|
| TRN01_RETIRO_EN_CUENTAS_PASIVAS | 87 | 32 | 2784 |
| TRN02_DEPOSITO_EN_CUENTAS_PASIVAS | 67 | 32 | 2144 |
| TRN03_ALTA_DPF | 140 | 10 | 1400 |
| TRN04_CONFIRMACION_DE_ALTA_DPF | 27 | 32 | 864 |
| TRN11_COMPRA_DE_MONEDA_EXTRANJERA | 27 | 32 | 864 |
| TRN12_VENTA_DE_MONEDA_ENTRANJERA | 27 | 32 | 864 |
| **Total** | **375** | | |

Debido a que el anterior escenario no es fielmente representativo de realidad, se decidió extrapolar el 10% sólo a aquellas transacciones incluidas en la prueba. Es importante aclarar que siempre se trata de una simplificación de la realidad ya que no es posible en automatizar todas las funcionalidades y casuísticas que se utilizan en la operativa diaria. En la siguiente tabla se observa el escenario del 10% de la carga, extrapolado para el subconjunto de transacciones antes detallado.

| Transacción | Usuarios concurrentes | Iteraciones (Por Usuario Por hora) | Total de datos a procesar |
|---|---|---|---|
| TRN01_RETIRO_EN_CUENTAS_PASIVAS | 41 | 32 | 1312 |
| TRN02_DEPOSITO_EN_CUENTAS_PASIVAS | 31 | 32 | 992 |
| TRN03_ALTA_DPF | 71 | 10 | 710 |
| TRN04_CONFIRMACION_DE_ALTA_DPF | 11 | 32 | 352 |
| TRN11_COMPRA_DE_MONEDA_EXTRANJERA | 11 | 32 | 352 |
| TRN12_VENTA_DE_MONEDA_ENTRANJERA | 11 | 32 | 352 |
| **Total** | **176** | | |

En esta sección se presentan, a modo de ejemplo, algunas de las ejecuciones asociadas al 10% de la carga esperada (tabla anterior).

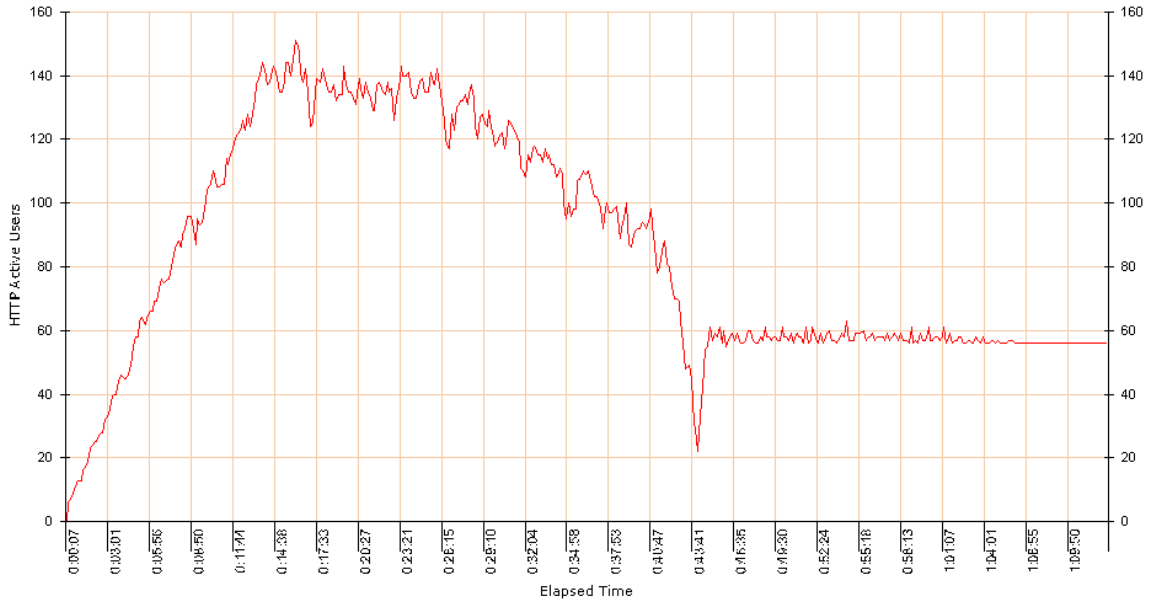
Las ejecuciones iniciales permitieron, brindar una primera visión de cómo se comportaban ante concurrencia tanto los scripts como el sistema, e identificar algunas condiciones en los datos que se fueron depurando. Se realizaron varias ejecuciones de este escenario de prueba.

En las siguientes secciones se detallan cronológicamente algunas de las pruebas ejecutadas para distintas cargas mostrando algunos elementos que ilustran los resultados obtenidos en cuanto a tiempos de respuesta y monitorización de la infraestructura durante la prueba.

Como se mencionó anteriormente, esta prueba, es similar a las que siguen en esta sección, corresponden al 10% de la carga esperada extrapolada para las seis transacciones, lo cual equivale a un total de 180 usuarios concurrentes aproximadamente. La prueba duró, en el entorno, de una hora.

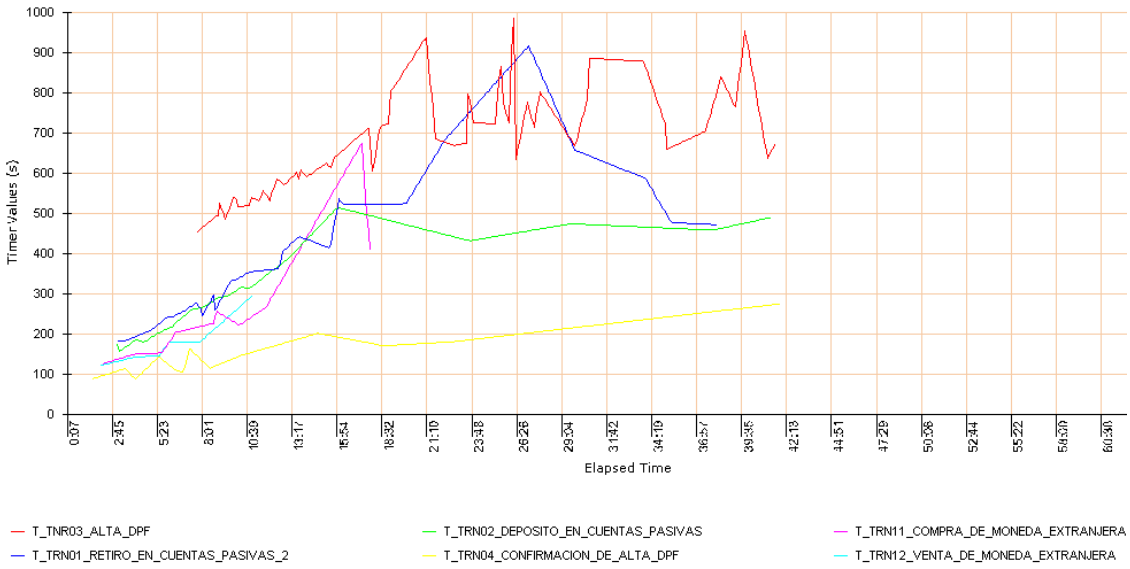
La "Gráfica 2" muestra la evolución de la cantidad de usuarios activos durante la prueba. Se observan cinco faces: *ramp-up* (primeros 15 minutos), un periodo "estacionario" (minuto 15 al 30), *ramp-down* (minuto 30 al 45), "estacionario posterior" (45 a 1:05) y "constante" (1:05 al final). Lo que se busca en la simulación es tener *ramp-up* para ir estudiando la sensibilidad a la concurrencia, el "estacionario" donde se estudia que soporte la carga durante un lapso representativo, el *ramp-down* es el efecto que logra el *ramp-up* el que ingresa primero terminará primero y es de esperar que si no hubieron problemas y se configura la misma cantidad de iteraciones para los usuarios de una misma transacción dure lo mismo que el *ramp-up.* El resto de las faces no se buscan intencionalmente y pueden darse por características del escenario, errores en la configuración y particularmente la etapa "constante" muestra que algún problema hizo que algún script quede colgado y no finalice su ejecución.

El eje horizontal de la Gráfica 2 representa el tiempo de prueba en minutos y en el eje vertical la cantidad de usuarios activos. El pico de concurrencia se da al terminar el *ramp-up* con aproximadamente 150 usuarios concurrentes (solicitudes http a la vez).

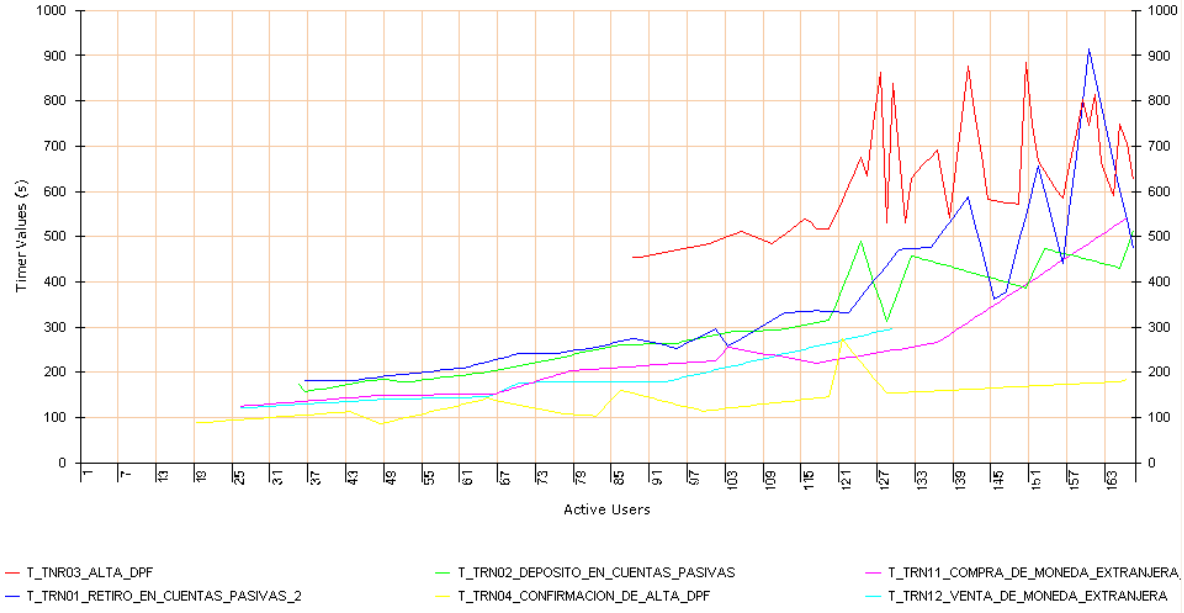



**Gráfica 2: Cantidad de usuarios durante el período de prueba**

Los tiempos de respuesta se muestran en las siguientes gráficas. El eje vertical representa el tiempo de respuesta en segundos. El eje horizontal de *Gráfica 3* es el largo de la prueba en minutos. El eje horizontal de la *Gráfica 4* representa la cantidad de usuarios activos (usuarios virtuales ejecutando).




**Gráfica 3: Tiempos de respuesta en función del tiempo de ejecución de prueba**




**Gráfica 4: Tiempos de respuesta en función de la cantidad de usuarios activos durante la prueba**

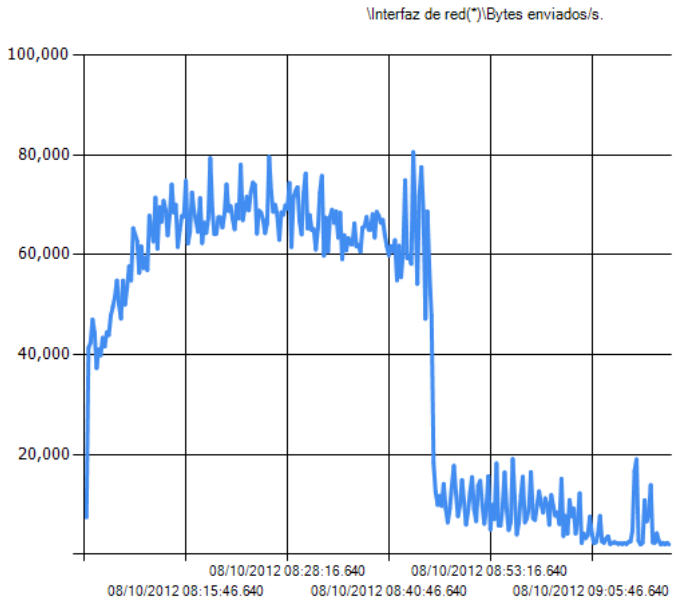
Se observa que para todas las transacciones los tiempos de respuesta se incrementan con el tiempo y a medida que ingresan más usuarios a la prueba, alcanzando tiempos altos de hasta 900 segundos (incluyendo *think times*), inaceptables para la operativa del sistema.

En la siguiente tabla se observa un resumen de los resultados obtenidos en segundos para las diferentes transacciones incluidas en la prueba.

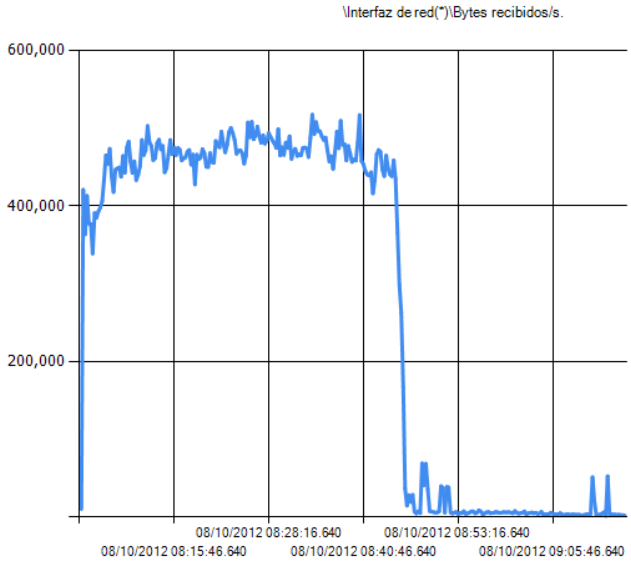
| ***Transacción*** | ***Promedio BaseLine*** | ***Mínimos*** | ***Máximos*** | ***Promedios*** | ***Percentiles 90*** |
|---|---|---|---|---|---|
| TRN01_RETIRO_EN_CUENTAS_PASIVAS | 85.9 | 180.84 | 915.98 | 385.12 | 586.82 |
| TRN02_DEPOSITO_EN_CUENTAS_PASIVAS | 82.4 | 156.92 | 514.18 | 311.36 | 478.40 |
| TRN03_ALTA_DPF | 343.5 | 453.17 | 801.73 | 584.85 | 710.6 |
| TRN04_CONFIRMACION_DE_ALTA_DPF | 51.8 | 86.82 | 274.2 | 146.12 | 194.579 |
| TRN11_COMPRA_DE_MONEDA_EXTRANJERA | 57.7 | 125.45 | 672.43 | 267.95 | 434.947 |
| TRN12_VENTA_DE_MONEDA_EXTRANJERA | 70 | 121.68 | 296.89 | 177.02 | 238.085 |

Durante la ejecución se detecta que el programa “HCLT01352” entra en *loop*. En el momento de ejecución quedan al menos 10 requerimientos ejecutando un número muy grande de sentencias por segundo, lo cual consume recursos considerablemente e invalidaba los resultados de la prueba. En particular, el problema se corresponde con la “TRN01 de Retiro en Efectivo”, la cual para ciertos datos específicos al ser ejecutada incluso desde la interfaz de la aplicación deja al sistema “colgado” procesando por varios minutos y finalmente arroja un error de servidor.

La monitorización de la generadora de carga muestra resultados normales para los contadores correspondientes a uso de procesador y uso de memoria. Sin embargo, para el uso de la red, en particular para la cantidad de bytes enviados y recibidos, los resultados parecen indicar que solo la generación de carga está consumiendo todo el ancho de banda del canal. Lo anterior se visualiza en la *Gráfica 5* y *Gráfica 6*.




**Gráfica 5: Bytes enviados por segundo**




**Gráfica 6: Bytes recibidos por segundo**

Se observa que durante el período de prueba la cantidad de bytes por segundo enviados es aproximadamente de 65KB/s, mientras que para los recibidos es de 450KB/s. En total, el ancho de banda utilizado es de aproximadamente 515KB/s, lo cual es el tope establecido por el enlace utilizado para la comunicación con el sistema (4Mb/s = 512KB/s). Para futuras pruebas se configura monitoreo del enlace al centro de cómputos para verificar si efectivamente ahí está el cuello de botella que genera tiempos inaceptables.

Luego de varias pruebas se logró solucionar el problema del programa que quedaba en *loop*. En las siguientes pruebas ejecutadas, que se detallan de aquí en más, el sistema cuenta con la corrección al problema.

### 16/08/2012

Luego de cada cambio, que supuestamente mejora la performance del sistema, se realiza una prueba con el fin de verificar su impacto. La prueba que se detalla a continuación es la que verifica la solución al problema referente al programa "HCLT0135" que quedaba en *loop* consumiendo una gran cantidad de recursos del sistema. Por lo tanto, esta prueba se corresponde al mismo escenarios de carga que los anteriores.

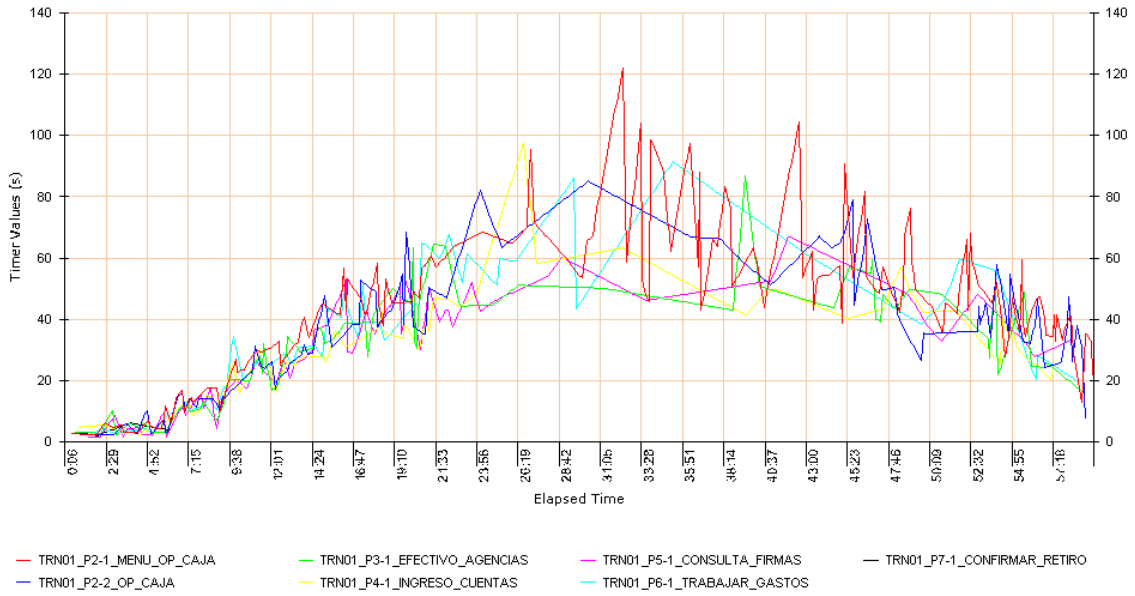
En esta prueba los tiempos de respuesta mejores, en comparación a pruebas anteriores de esta carga, se supone debido al *bug* corregido. Sin embargo siguieron incrementándose con la cantidad de usuarios. Esto indica que aún existe un problema que hace que los tiempos obtenidos no sean aceptables.

La monitorización configurada hasta el momento para los diferentes componentes de la infraestructura, no permitía evidenciar el cuello de botella que causaba los tiempos altos.

### 22/08/2012

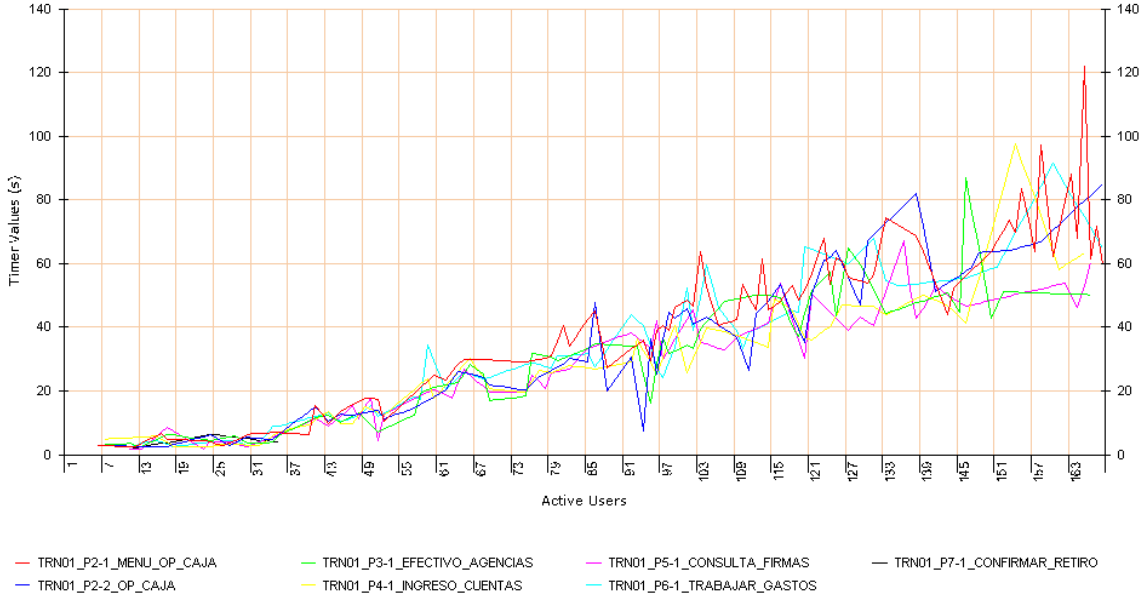
Las pruebas anteriores motivaban la ejecución de una prueba análoga, pero poniendo especial énfasis en la monitorización del canal al centro de cómputos que es el componente para el cual aún no se lograba acceder a información de monitorización, con el objetivo de verificar si éste estaba siendo el cuello de botella responsable de que los tiempos de respuesta fueran altos durante las pruebas ejecutadas anteriormente.

En una prueba de performance los tiempos pueden aumentar para todo el sistema o para determinadas funcionalidades o pantallas. El análisis de las transacciones en pasos permite detectar patrones. Si los tiempos de respuesta aumenta para todos los *timers* de forma pareja reafirma la hipótesis de un cuello de botella a nivel de recursos, en caso de detectar tiempos altos para algunas transacciones o solo en algunos pasos de las transacciones se puede estudiar de forma aislada lo "problemático". La gráficas a continuación muestran los *timers* que ejecutaron correctamente de la "TRN01 – Retiro". Particularmente la siguiente gráfica confirma que todos se degradan de forma pareja y que al final de la prueba, a medida que los usuarios completan la misma y hay menos usuarios (correlacionarla con la Gráfica 9), los tiempos mejoran.

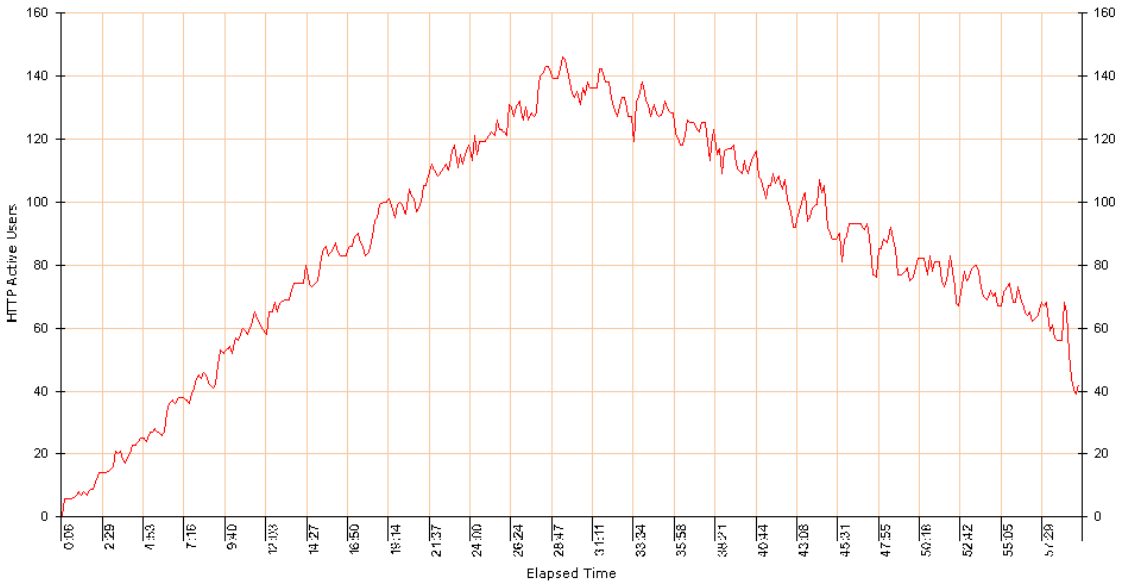



**Gráfica 7: Tiempos de respuesta en función del tiempo de ejecución de prueba**

Analizando la siguiente gráfica confirmamos que la degradación a lo largo de la prueba se da a medida que ingresan más usuarios, comportamiento que se da para todos los pasos de la transacción. Esta gráfica permite rápidamente identificar los límites en lo que respecta a cantidad de usuarios activos donde el sistema se empieza a degradar, en este caso con 35 usuarios aproximadamente se detecta el inicio de una rampa, el estudio junto a la gráfica anterior es importante para entender que el efecto se da por la cantidad de usuarios y no por el largo de la prueba (no confundir con el anti-patrón “rampa” citado más adelante).




**Gráfica 8: Tiempos de respuesta en función de la cantidad de usuarios activos durante la prueba**

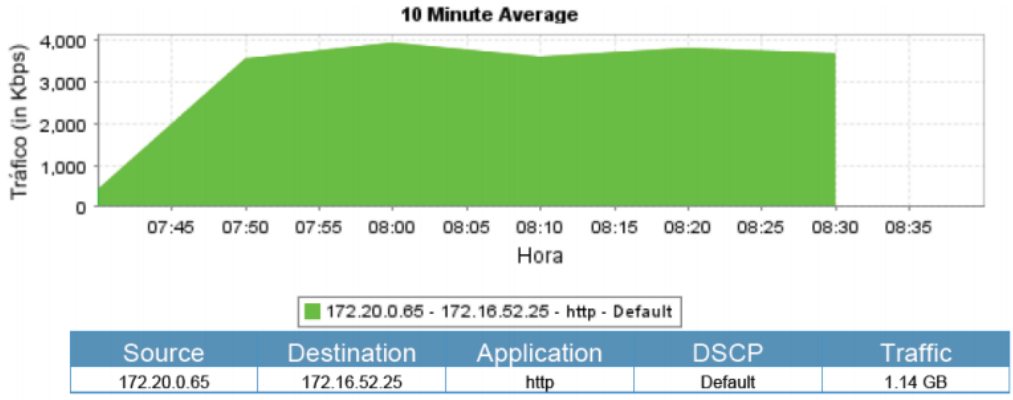



**Gráfica 9: Cantidad de usuarios activos durante el período de prueba**

Un comportamiento similar al anteriormente descripto se da para todas las demás transacciones.

Tener en cuenta que los tiempos incluidos en las gráficas no incluyen *think times*, por lo que son todos tiempos de procesamiento de servidor o eventualmente tiempos de red.

Por otro lado, en la *Gráfica 10* se muestra el consumo del canal del centro de cómputos.




| Source | Destination | Application | DSCP | Traffic |
|---|---|---|---|---|
| 172.20.0.65 | 172.16.52.25 | http | Default | 1.14 GB |

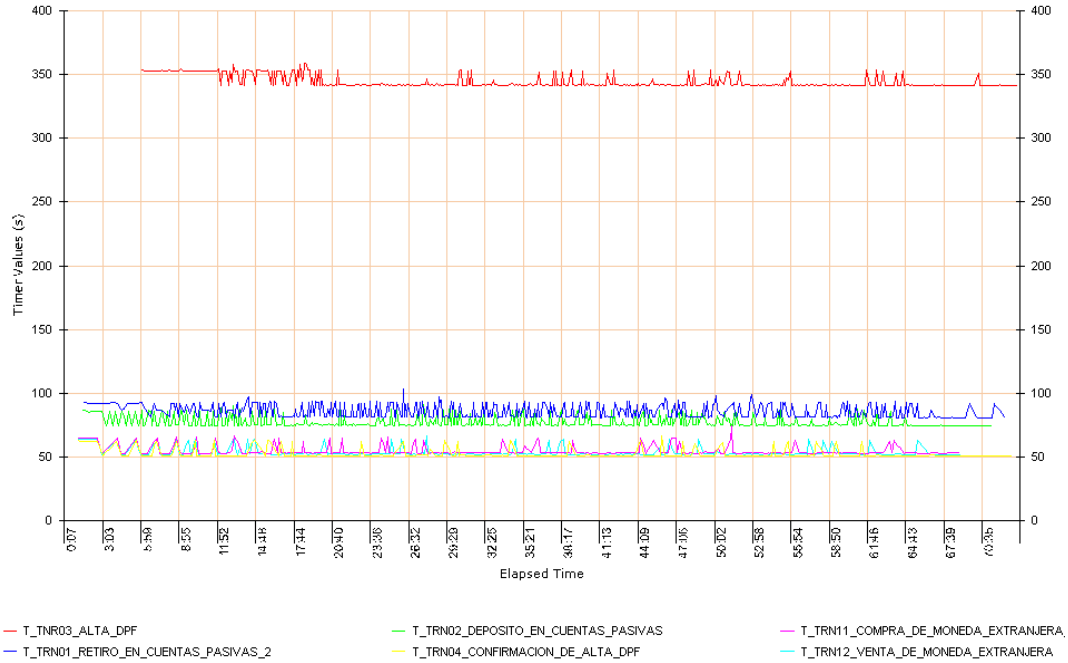
**Gráfica 10: Consumo del canal al centro de cómputos durante la prueba**

Se observa que el tráfico está casi en el máximo posible ya que el enlace es de 4Mbps. Lo anterior es un posible indicio de que el cuello de botella que hace que los tiempos se incrementen es la red, en particular el enlace que comunica las generadoras de carga con el centro de cómputos, por lo que de cara a las siguientes pruebas se decide conectar las generadoras de carga directamente en el centro de cómputos para mitigar el problema.

Los resultados de la monitorización para la generadora de carga son similares a los ya presentados para las pruebas anteriores, observando que la generación de carga es soportada por el equipo, pero el uso de ancho de banda está muy cerca del tope establecido por el enlace disponible, como también se pudo apreciar en la monitorización correspondiente al uso del canal entre la generadora y el centro de cómputos donde está alojado el SUT.

### 30/08/2012

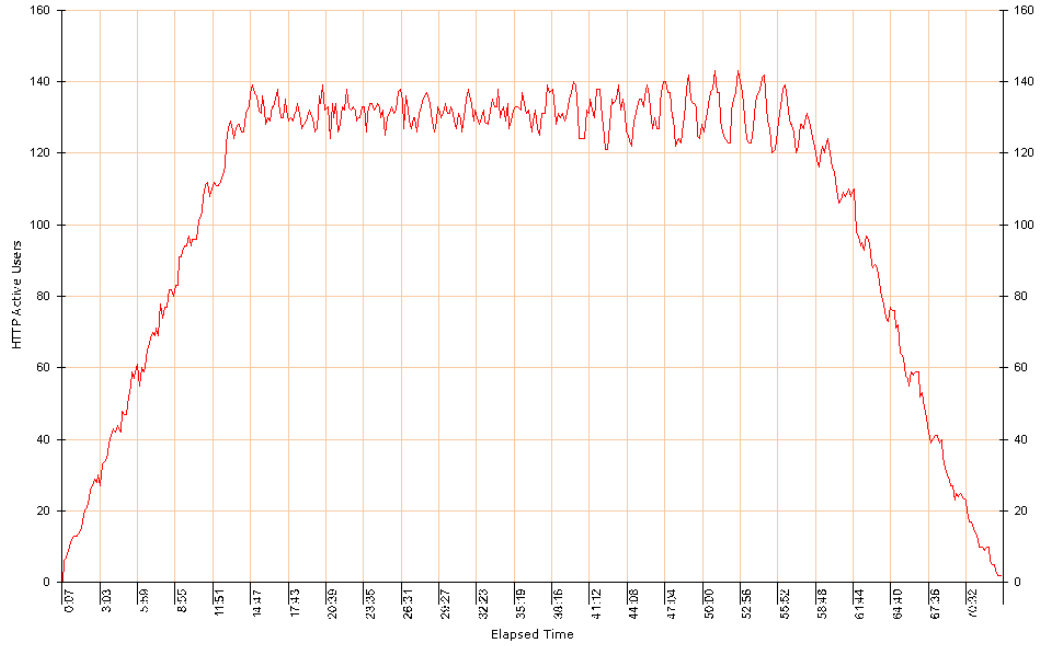
Luego de la migración de las generadoras al centro de cómputos se ejecutó una prueba para confirmar que no había otro cuello de botella. Los resultados obtenidos se pueden observar en la *Gráfica 11*.




**Gráfica 11: Tiempos de respuesta en función del tiempo de ejecución de prueba**

Como se observa, los resultados fueron positivos en cuanto a tiempos de respuesta, ya que se mantuvieron en valores aceptables (similares a los tiempos de base) durante el transcurso de la prueba. Se logró comprobar que el cuello de botella efectivamente se encontraba en el enlace que comunicaba la generadora con el centro de cómputos.

En la *Gráfica 12* se muestra la evolución de usuarios activos durante la prueba.




**Gráfica 12: Cantidad de usuarios activos durante el período de prueba**

Se observa que se alcanzó un pico de aproximadamente 140 usuarios activos, aunque la carga se correspondió a 180 usuarios que realizaron el *login* en la aplicación.

Para esta prueba la monitorización arrojó resultados positivos en cuanto al uso de recursos por parte de la generadora.

#### 3.4.2.3. 50% de la carga

Dados los resultados de la prueba anterior, se decidió continuar incrementando la carga a un 50% de la carga esperada, lo cual equivale aproximadamente a 850 usuarios. Como se observa en la siguiente tabla la prueba se basó en las seis transacciones incluidas en las pruebas anteriores.

| Transacción | Usuarios concurrentes | Iteraciones (Por Usuario Por hora) | Total de datos a procesar |
|---|---|---|---|
| TRN01_RETIRO_EN_CUENTAS_PASIVAS | 205 | 32 | 6560 |
| TRN02_DEPOSITO_EN_CUENTAS_PASIVAS | 155 | 32 | 4960 |
| TRN03_ALTA_DPF | 355 | 10 | 3550 |
| TRN04_CONFIRMACION_DE_ALTA_DPF | 55 | 32 | 1760 |
| TRN11_COMPRA_DE_MONEDA_EXTRANJERA | 55 | 32 | 1760 |
| TRN12_VENTA_DE_MONEDA_ENTRANJERA | 55 | 32 | 1760 |
| **Total** | **880** | | |

Los resultados obtenidos para estas pruebas también permanecieron estables durante toda la hora de ejecución y se consideran aceptables en comparación con los tiempos de base.

Los resultados de la monitorización para la generadora de carga muestran que la generación es soportada por el equipo. Al contar con un enlace de 10 Gb/s la red dejó de ser el cuello de botella, el ancho de banda es suficiente para ejecutar la carga sin inconvenientes. En este caso, durante la prueba se envían datos aproximadamente a una tasa de 400 kb/s y se reciben datos a una tasa aproximada de 3.8 Mb/s.

#### 3.4.2.4. 100% de carga

##### 30/08/2012 - Prueba de larga duración

La prueba que se detalla en esta sección tiene mayor duración (nueve horas) que las que se venían realizando (una hora). Extendiendo el periodo de la prueba y manteniendo el *throughput* de carga se pueden detectar errores que se hacen visibles con el "envejecimiento", por ejemplo fugas de memoria y que evidencian el anti-patrón de performance "La rampa" ("*The ramp*") (Smith & Williams, New Software Performance AntiPatterns: More Ways to Shoot Yourself in the Foot). El *ramp-up* configurado fue de dos horas para que en ese lapso ingresen casi 1800 usuarios simulados al sistema.

Los resultados fueron positivos en cuanto a los tiempos de respuesta, ya que los mismos se mantuvieron en valores aceptables (similares a los tiempos de base) y fueron estables durante la prueba.

##### 19/09/12

Se ejecutó otra prueba correspondiente al 100% dado que al ejecutar el 200% (Ver prueba "18/09/12 – 200%") integrando la transacción seis, los tiempos se degradaron resultando inaceptables para la operativa. La transacción seis tiene la particularidad que cientos de usuarios la ejecutan pero una sola vez. Su criticidad viene dada por el volumen de usuarios (sesiones) que agrega al SUT. Se decidió disminuir la carga, con respecto a esa prueba anterior, a fin de verificar si el 100% de la carga era soportado con las nueva transacción.

El escenario ejecutado en este caso se detalla en la siguiente tabla.

| Transacción | Usuarios concurrentes | Iteraciones (Por Usuario Por hora) | Total de datos a procesar |
|---|---|---|---|
| TRN01_RETIRO_EN_CUENTAS_PASIVAS | 41 | 32 | 1312 |
| TRN02_DEPOSITO_EN_CUENTAS_PASIVAS | 31 | 32 | 992 |
| TRN03_ALTA_DPF | 71 | 10 | 710 |
| TRN04_CONFIRMACION_DE_ALTA_DPF | 11 | 32 | 352 |
| TRN06_SOLICITUD | 6126 | 1 | 6126 |
| TRN11_COMPRA_DE_MONEDA_EXTRANJERA | 11 | 32 | 352 |
| TRN12_VENTA_DE_MONEDA_ENTRANJERA | 11 | 32 | 352 |
| **Total** | **6302** | | |

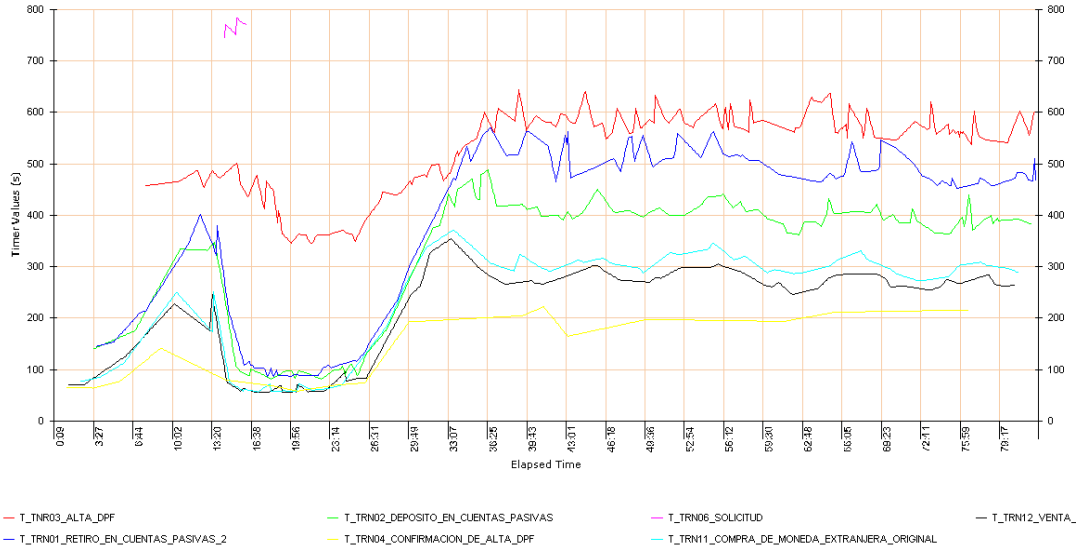
Los tiempos de respuesta obtenidos para esta prueba se mantuvieron estables durante toda la ejecución. En este caso, el pico de usuarios activos que se logró alcanzar fue de 550 usuarios aproximadamente.

El 100% de la carga no presentó problemas de performance, aun cuando se incluían la mayoría de las transacciones en la prueba, esto fundamento ejecutar nuevos incrementos de carga, el cual se detallará más adelante (Ver prueba “20/09/12 - 150% de la carga”).

### 3.4.2.5. Cargas mayores al 100% de la carga esperada

#### 18/09/12 – 200%

Previamente, a esta ejecución, se había ejecutado el 100% sin problemas (aunque con la parcialidad de las transacciones). Los resultados obtenidos se pueden visualizar en la *Gráfica 13* (los tiempos de respuesta incluyen *think times*).




**Gráfica 13: Tiempos de respuesta en función del tiempo de ejecución de prueba**

Se observa un incremento en los tiempos de respuesta para todas las transacciones a los pocos minutos de comenzada la prueba, lo cual se puede atribuir a la ejecución de dos procesos adicionales lanzados por usuarios que fueron cancelados una vez detectado el incremento de los

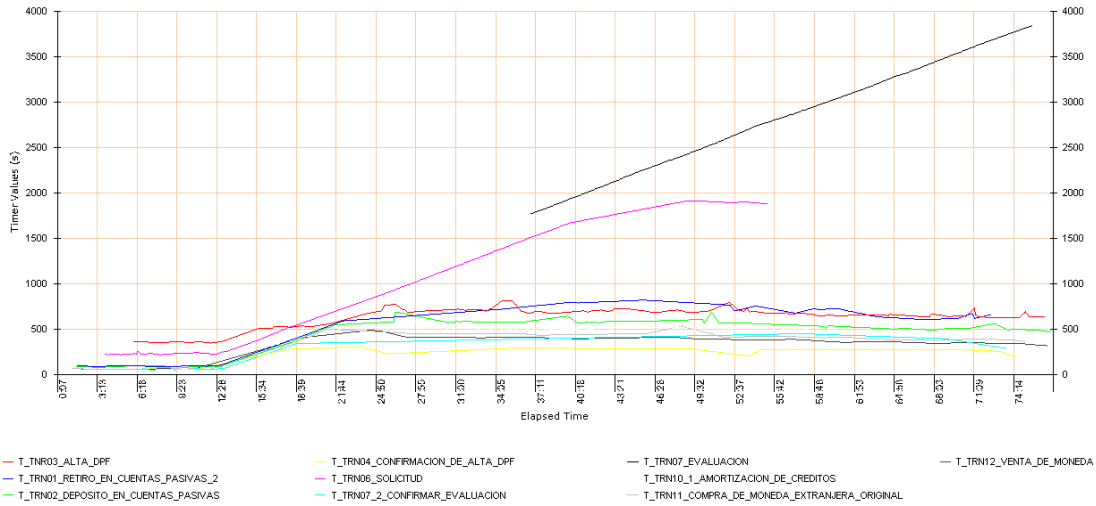
*commit* en el servidor. Una vez finalizados estos procesos los tiempos se estabilizan pero a mitad de la prueba vuelven a incrementarse estabilizándose pero sin recuperarse.

Para esta prueba se detectaron varios problemas y se tomaron algunas acciones:

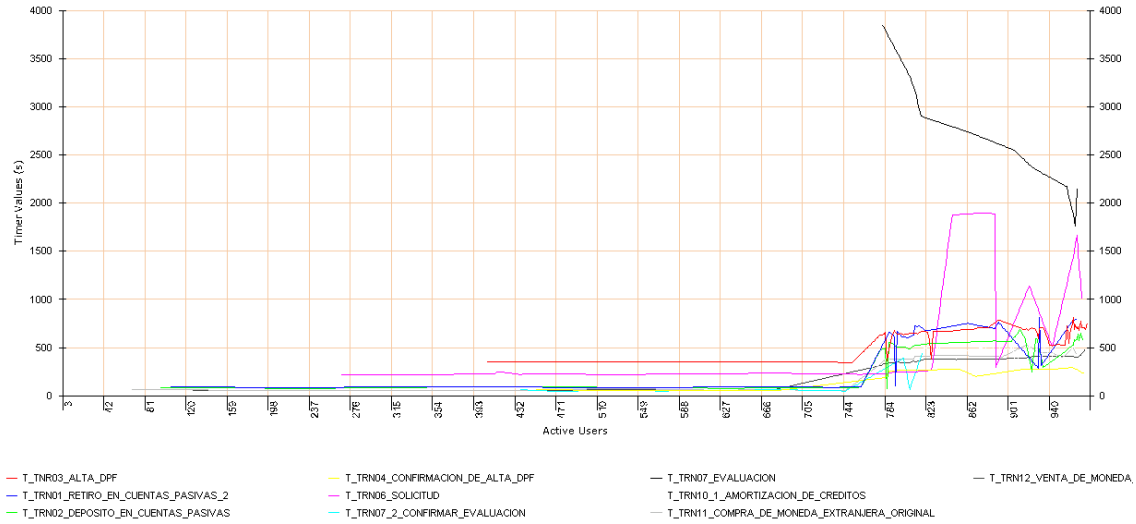
- Dada la carga observada en el *pool* y los usuarios que quedaban en espera por conexiones libres, se aumentaron la cantidad de conexiones (de 10 a 25). Algunas consideraciones:
    - El cambio se realizó en caliente
        - No queda grabado si se reinicia el ambiente hay que cambiarlo de nuevo.
    - No es definitivo. Es recomendable continuar monitoreando y ajustando.
    - Es probable que el número final ronde las 15
        - Considerando que el uso del cache aún no se había sido implantado.
- El programa que generó más carga fue "asngfcsg", que de acuerdo a las estadísticas ejecuta aproximadamente diez veces más sentencias SQL que el siguiente en la lista.
- El programa "hpop118" (un *popup*, posiblemente para seleccionar una cuenta o similar) continúa ejecutando una sentencia SQL que devuelve potencialmente varios miles de registros, aunque el mismo solo lee 10 para mostrarlos a pantalla, seguramente está faltando un "top".

## 20/09/12 - 150% de la carga

Los resultados obtenidos en este caso se visualizan en las Gráfica 14 y Gráfica 15. Nuevamente, tener en cuenta que los tiempos que se visualizan en los gráficos incluyen *think times*.




**Gráfica 14 - Tiempos de respuesta en función del tiempo de ejecución de prueba**




**Gráfica 15 - Tiempos de respuesta en función de la cantidad de usuarios activos durante la prueba**

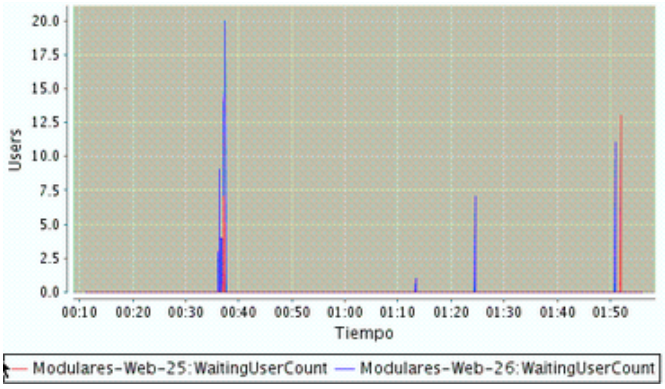
En este caso se observa que luego de los 12 minutos de prueba aproximadamente, los tiempos comienzan a degradarse, superando en el peor de los casos los 3500 segundos. La cantidad de usuarios activos a la cual se llegó como pico de carga es cercano a los 950 usuarios, pero según se observa en la Gráfica 15 el punto donde comienzan a incrementarse en mayor medida los tiempos es luego de los 750 usuarios activos aproximadamente.

Se verificó mediante un "usuario testigo" real que el SUT se percibía lento durante la carga.

### 20/09/12 - 200% de la carga

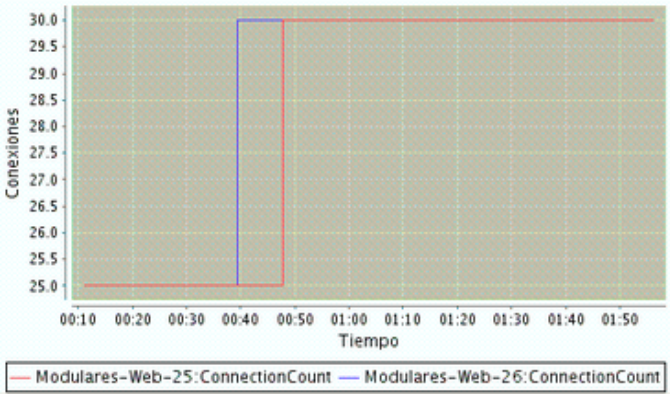
Esta prueba permitió documentar algunas de las acciones mencionadas anteriormente:

- Al inicio se detectaron esperas de los usuarios como se visualiza en la siguiente gráfica.



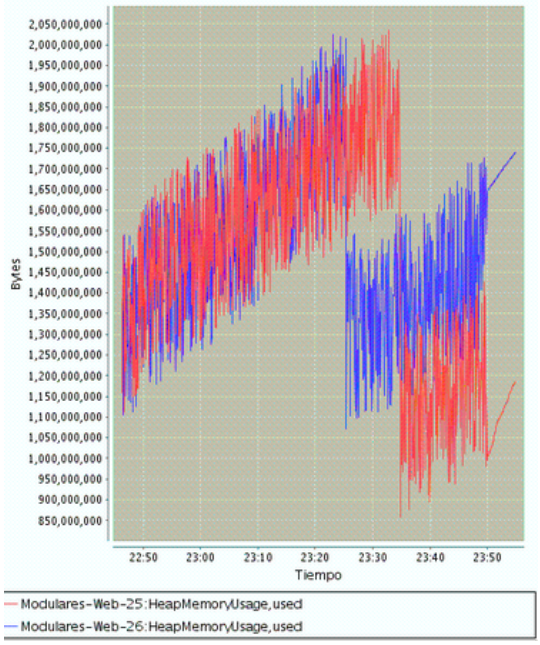

**Gráfica 16: Cantidad de usuarios en espera**

- Dada la carga observada y para evitar espera de los usuarios por conexiones libres, se aumentaron la cantidad de conexiones (de 25 a 30) en el *pool*.



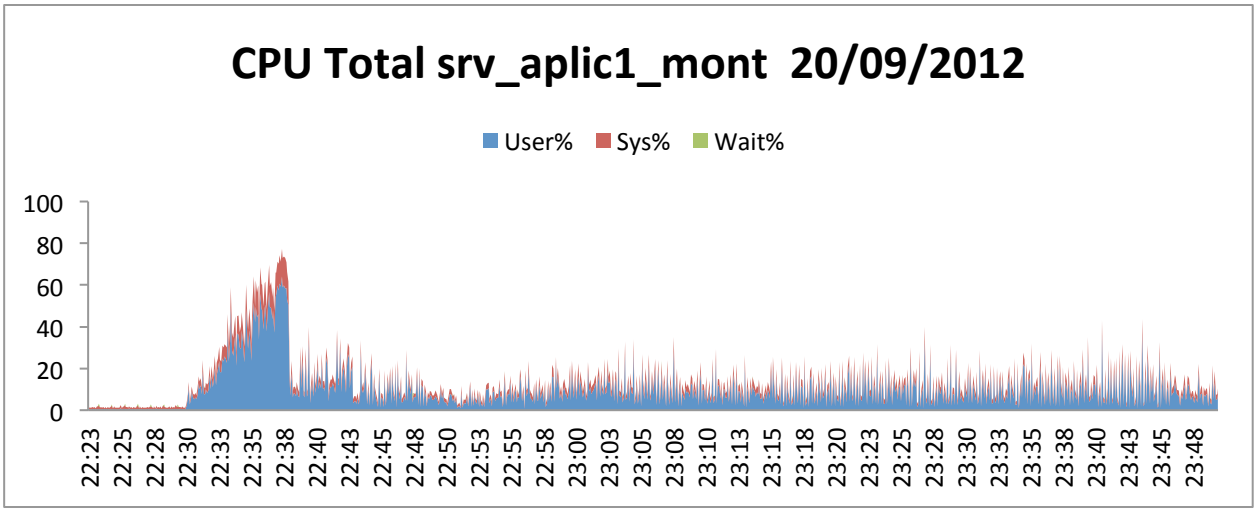

**Gráfica 17: Pool de conexiones**

- El *heap* de memoria se mantuvo sin problemas en el entorno de 1GB a 2GB, en la medida que se disparaban los *Garbage Collections.* La siguiente gráfica muestra un comportamiento es adecuado.




**Gráfica 18: Uso de memoria del *heap* de la JVM**

En la *Gráfica 19* se puede observar la actividad del WAS1, el cual no se ve exigido en cuanto al uso de CPU dado que llegaban muy pocos requerimientos. Para el WAS2 ocurre algo similar.




**Gráfica** 19**: Uso de CPU en WAS1**

Al no llegar la carga esperada a los WAS se diseñaron pruebas para verificar donde radicaba el problema. Estas pruebas se pueden ver a continuación.

### 01/10/12 – Direccionamiento del 150% de la carga

Con el objetivo de identificar la causa de los problemas de performance en pruebas anteriores. El día lunes 01/10/12 se ejecutaron tres pruebas dirigiendo el 150% la carga a distintos elementos:

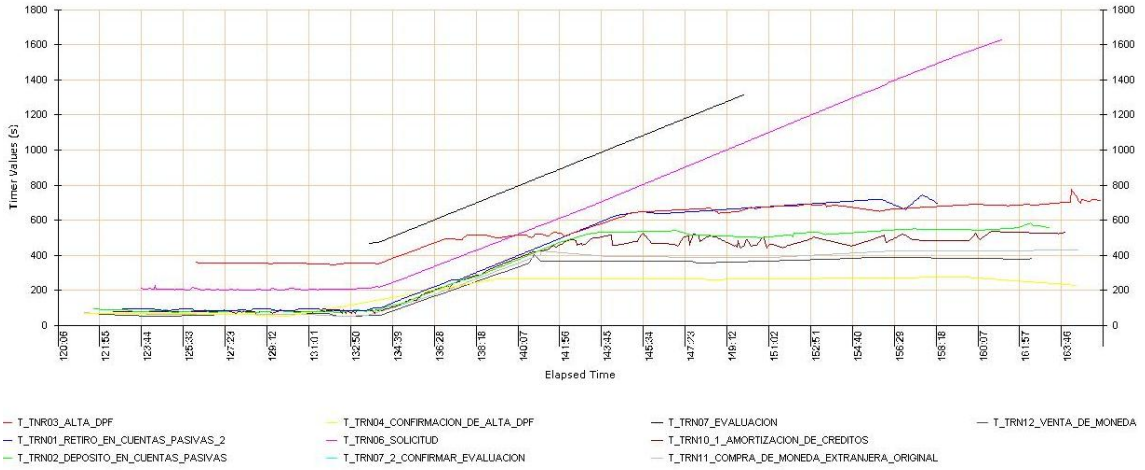
1. Hacia el *cluster*.
2. Hacia uno de los WAS.
3. Directa hacia uno de los nodos IBM HTTP Server.

Al momento de validar los scripts para la ejecución de estas pruebas, se observa que comienzan a fallar por temas asociados a la codificación de las respuestas. Personal de DL&A confirma que se agregó la compresión (gzip) a las respuestas, por lo que de aquí en adelante el tráfico viaja comprimido. Este cambio afecta las validaciones de los scripts.

Las pruebas que siguen se realizaron con el personal de DL&A físicamente en las oficinas del CES. A continuación se presentan los resultados.

**Prueba 1 - Carga de 150% hacia el *cluster***

En esta prueba, ocurrió lo que venía sucediendo en pruebas anteriores (de 150% de carga). Luego de 15 minutos de prueba, y al superar los 800 usuarios, los tiempos comienzan a degradarse. Este comportamiento se puede visualizar en la siguiente gráfica de tiempos de respuesta con *think times*.




**Gráfica** 20**: Tiempos de respuesta en función del tiempo de ejecución**

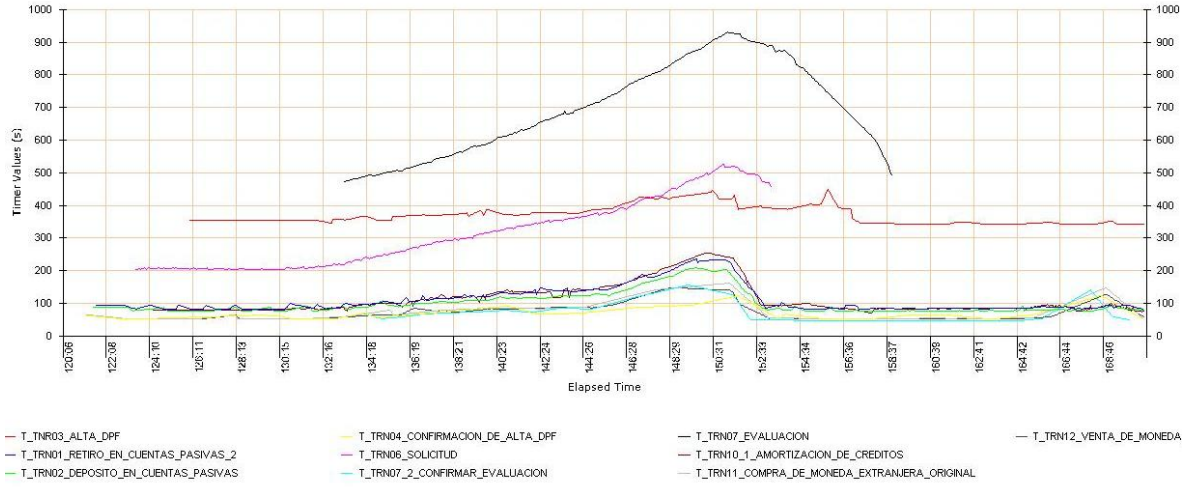
En definitiva, se repitió lo que ya se había observado, tiempos altos y carga normal del lado del WAS.

### Prueba 2 - Carga de 150% directa hacia uno de los WAS

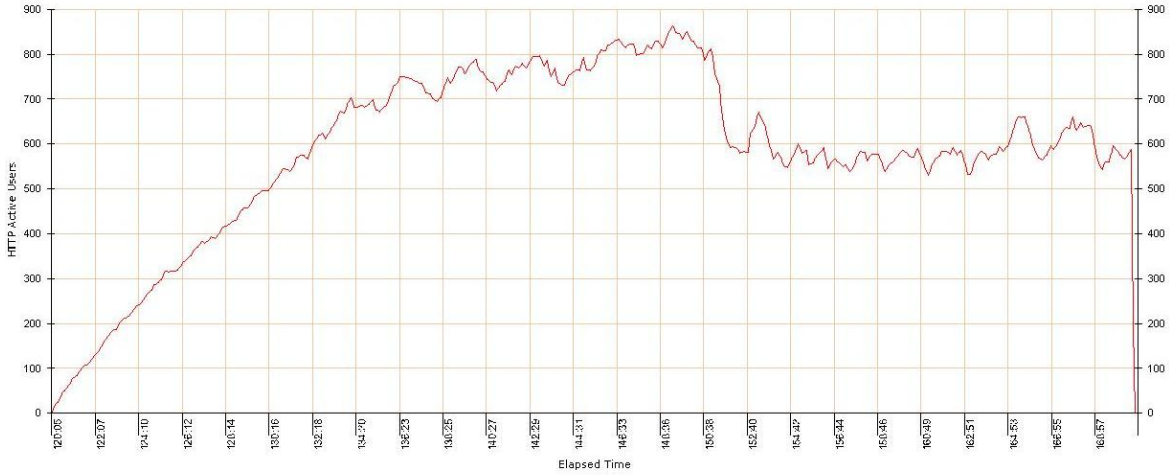
Esta prueba respondió mejor que la anterior, aunque de todas formas los tiempos se vieron degradados (hay que tener en cuenta que esta carga debería de distribuirse en dos servidores).

Si se correlacionan las siguientes gráficas y compara con la anterior, se puede observar que transcurridos los 30 minutos de  hay una caída algo abrupta de la cantidad de usuarios activos y luego de esto mejoran los tiempos de respuesta. Esto es debido a que en ese punto, se resetearon las conexiones del servidor para liberar memoria, lo que detuvo la actividad de unos 200 usuarios y a partir de aquí se estabilizó la prueba.

En definitiva, tomando en cuenta que el WAS estaba sobrecargado puesto estaba recibiendo la carga del 150% solo, en vez balanceada entre dos, los tiempos de respuesta fueron mejores que la prueba anterior.




**Gráfica 21: Tiempos de respuesta en función del tiempo de ejecución**




**Gráfica 22: Usuarios activos en función del tiempo transcurrido**

**Prueba 3 - Carga de 150% directa hacia uno de los nodos IBM HTTP Server**

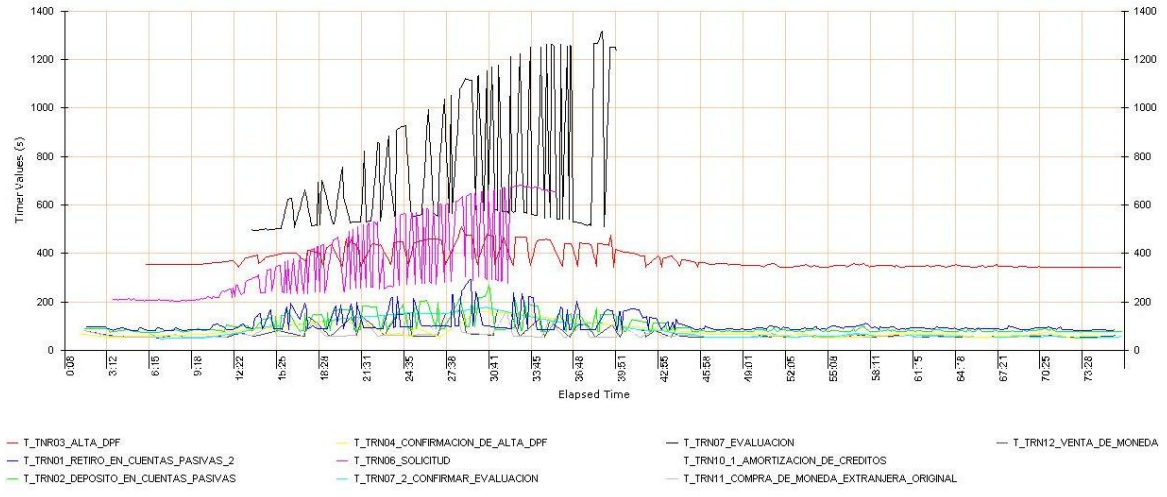
En esta prueba los tiempos de respuesta comenzaron a degradarse luego de cinco minutos de iniciada la prueba, con unos 300 usuarios. Si se comparan estas tres pruebas se puede decir que esta fue la que arrojó peores resultados.

En definitiva, se repitieron los problemas de la primer prueba para este mismo grupo de pruebas, pero potenciados, ya que la carga no se balancea sino que es recibida específicamente por un solo IBM HTTP Server.

## 03/10/2012 - 150% de carga

Luego de detectar y aplicar una posible solución al problema que limitaba la cantidad de conexiones en los IBM HTTP Server (conexiones simultaneas), el día miércoles 03/10/2012 se ejecutó una prueba correspondientes al 150% carga esperada con el fin de verificar que estos cambios realmente eran una solución.

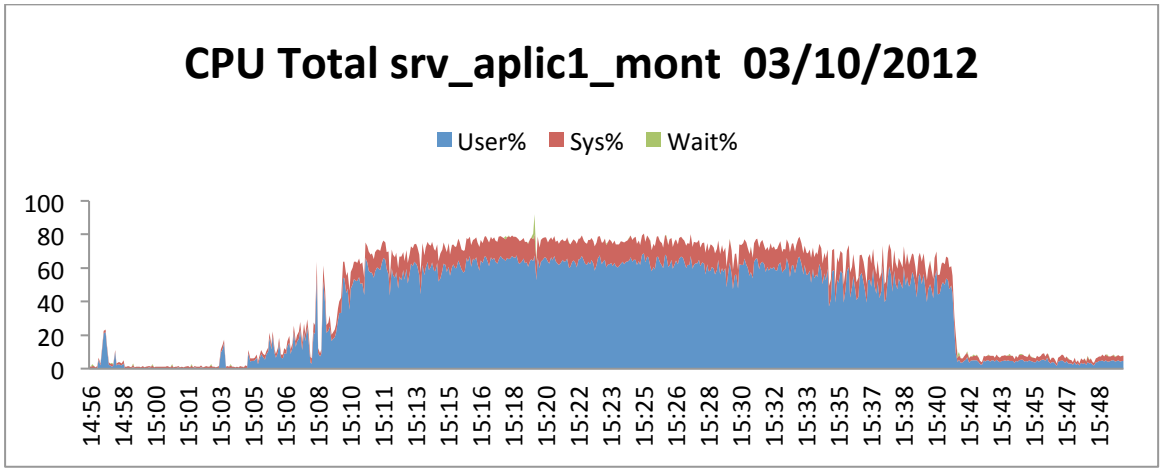
La prueba determinó un nuevo comportamiento “estilo sierra”. Dado que durante los primeros minutos de la prueba se activó un proceso en uno de los servidores WAS se procedió a realizar una segunda prueba con el fin de verificar si los tiempos altos se correspondían a un servidor (el cual ejecutaba el proceso). A continuación se presentan los resultados de la segunda prueba.




**Gráfica 23: Tiempos de respuesta en función del tiempo de ejecución**

En esta prueba se repite el comportamiento de la prueba anterior. Se puede ver cómo los tiempos de respuesta oscilan, y cómo se degradan con el tiempo. Se podría presumir que esto se debe a como se balanceó la carga. Otra teoría que se consideró fue que uno de los WAS se vio más cargado que el otro debido a que tenía menos memoria asignada.

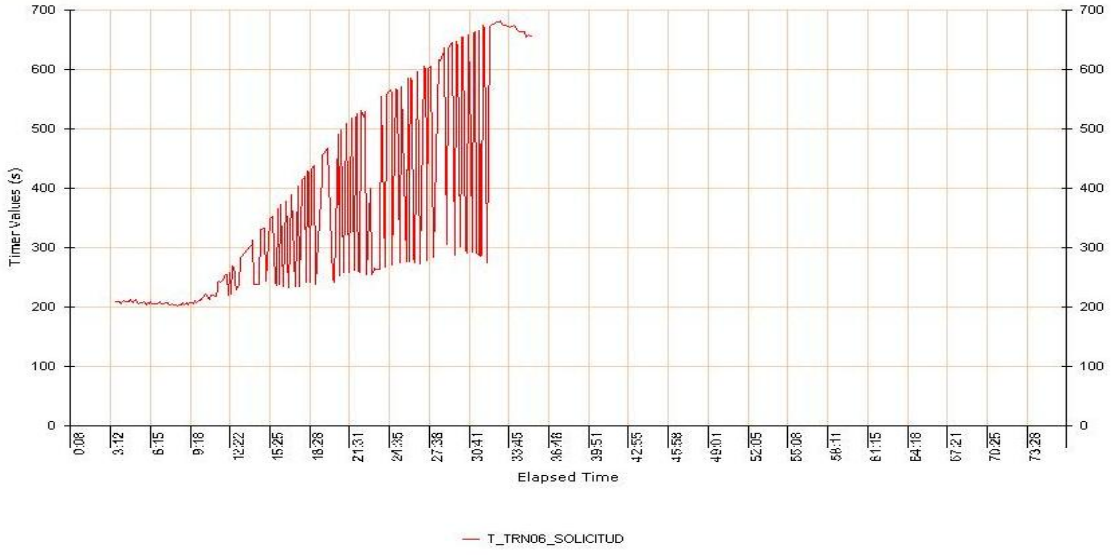
En la *Gráfica 24* se muestra el uso de CPU para el WAS1 (similar para WAS2). Si se lo compara con *Gráfica 19*, se puede ver que la exigencia de CPU para este caso es mucho mayor durante la prueba. Lo cual es un punto positivo ya que empieza a llegar carga a los WAS (presumiendo que se corrige el problema en la balanceadora que limitaba el número de conexiones y por lo tanto no dejaba pasar tráfico/carga).




**Gráfica 24: Uso de CPU en WAS1**

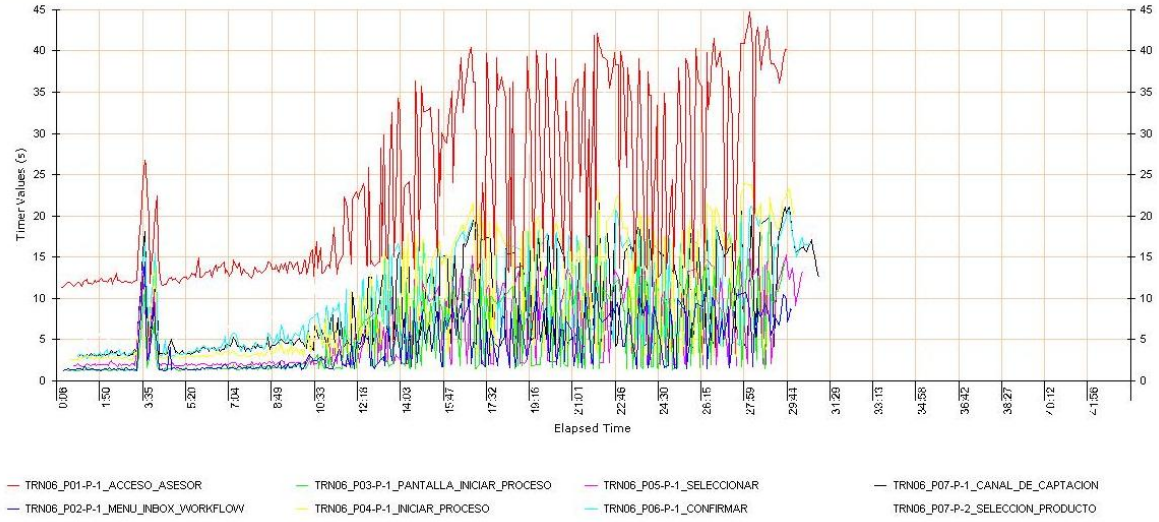
A continuación se muestra el comportamiento de la Solicitud de Crédito (transacción seis) y se presentan gráficas de tiempos de respuesta para cada uno de los pasos de dicha transacción.

En la siguiente gráfica se puede observar la evolución de los tiempos de respuesta que varían desde 200 hasta casi 700 segundos en las últimas muestras. Es importante tener en cuenta que el aumento de los tiempos de respuesta de una transacción es la sumatoria de la degradación del tiempo cada paso. Esta transacción tiene más de 20 pasos, con lo cual, seguramente los casi 500 segundos (que se degrada esta transacción) no sean de espera continua para el usuario final. El mismo comentario es importante tenerlo en cuenta al momento de analizar las la transacción siete que tiene más de 50 pasos.




**Gráfica 25: Tiempos de respuesta en función del tiempo transcurrido para TRN06**

En la siguiente gráfica se ven los tiempos de respuesta en función del largo de la prueba de los pasos uno al siete de la transacción seis.




**Gráfica 26: Tiempos de respuesta en función del tiempo transcurrido. Pasos uno a siete de la TRN06**

En esta gráfica, se puede ver como por ejemplo el paso 1 "Acceso Asesor" (el *login* de un asesor), lleva pasa de 10 segundos a 45. Por lo tanto el tiempo para completar esta transacción creció en más de 30 segundos.

También se ve como el resto de los tiempos de respuesta de los pasos se degradan, aunque en menor medida que el *login*. Esto muestra como se da el aumento del tiempo general mediante la suma de las demoras de los pasos. Observando globalmente vemos que los demás pasos se degradan entre 10 y 20 segundos cada uno.

Al analizar el resto de los *timers* se detectó que:

1. el paso 9-2, aumenta su tiempo en forma similar al *login*
2. los pasos 15-1, 17-1 y 22-1 aumentan su tiempo de respuesta en más de 20 segundos
3. el resto de los *timers* se degradan de 5 a 20 segundos por paso.

Se ha comprobado también durante las pruebas mediante usuarios testigos, que las operaciones de desplegar pop-ups con información tardan unos 10 segundos.

En este punto podríamos decir que los tiempos han mejorado, sin embargo siguen existiendo tiempos altos. Además apareció el nuevo problema de las gráficas oscilantes que degradan sus tiempos a lo largo de la prueba, sobre todo los de la envolvente superior (picos).

### 05/10/12 - 150% de carga

Dado que se sospechaba que el problema era causado por la balanceadora la cual daba más carga a un WAS que al otro, el día viernes 05/10/2012 se realizó una nueva prueba del 150% de la carga esperada repartida en proporciones similares entre los dos WAS. El objetivo de esta prueba era seguir acotando el problema con lo cual:

- Si uno de los WAS no soportaba la carga el problema apuntaba a ese server.
- Si ninguno de los WAS soporta el problema que generaba los tiempos oscilantes de la última prueba era que uno estaba liviano y el otro sobrecargado.
- Si la carga era soportada por ambos WAS el problema apuntaba a la balanceadora.
- Finalmente, si los tiempos figuran oscilantes en ambas configuraciones la causa podría ser los datos.

La configuración fue la siguiente:

- 75% de carga desde tres generadoras de carga (*.91, *.92, *.93) directa a un WAS (*.25)
- 75% de carga desde tres generadoras de carga (*.94, *.95, *.96) directa a un WAS (*.26)

A continuación se muestran los gráficos resultantes de la prueba.

**Gráficas para el WAS (*.25)**

Se observa que:

1. Los tiempos se degradaron de forma irregular, oscilando aunque no tanto como en las pruebas sobre la balanceadora.
2. En el WAS *.25 no se degrada como el resto, al principio, pero luego de un tiempo empiezan a ocurrir errores (desde el minuto 35 en adelante).
3. Durante la prueba ocurrió que ambos WAS hicieron "*reload*", lo que produjo una caída abrupta de la cantidad de usuarios activos.

### 08/10/12 - 150% de carga

El día lunes 08/10/2012 fueron reiniciados los servidores y se realizaron dos pruebas del 150% de la carga esperada distribuida de forma diferente, la segunda prueba se realizó debido a los resultados arrojados por la primera.

Se reinicia el servidor Power 740 donde están alojados virtualmente todos los componentes de la infraestructura y se ejecuta una prueba similar a la anterior (apuntando a los WAS 75% y 75%), con el fin de verificar si la carga se “empareja”.

Para esta primera prueba la distribución fue como se había planificado, y la carga ejecutada por los WAS fue similar.

Esta prueba arrojó resultados aceptables en ambas generaciones del 75% con lo cual estas pruebas permitieron perfilar el análisis del problema hacia la configuración de la balanceadora.

Dado el resultado se paso a realizar una segunda prueba la cual consistió del 150% de la carga sobre la balanceadora. Antes de esta prueba, DLYA modificó un módulo de *debugging*, lo cual hizo que fallaran las transacciones 1, 2 y 10.

Se verificó, nuevamente, el comportamiento inestable “estilo sierra”, marcándose dos tendencias bastante claras para los valles y los picos (y sobre todo para las transacciones seis y siete).

Teniendo en cuenta que durante la prueba anterior no se detectó un comportamiento similar se atribuye, en principio, la diferencia de conducta a la balanceadora. El balanceo no funcionó bien. Uno de los server (el 26) tuvo más carga que el otro (del orden de un 30% aproximadamente). Personal de DL&A solicita que se revea el modo de balanceo que están realizando los equipos dedicados a esa tarea.

### 09/10/12

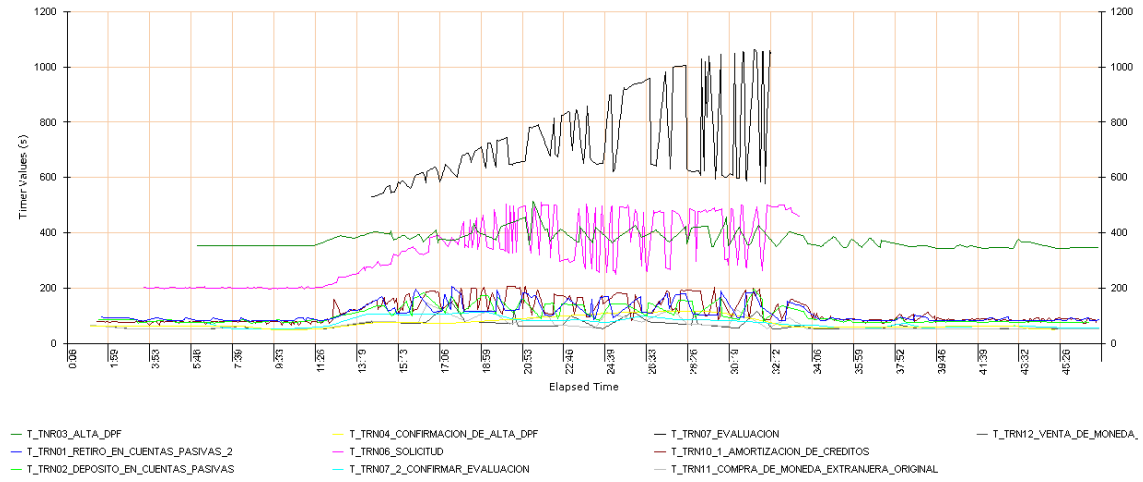
Se ejecutaron dos nuevas pruebas luego de nivelar la asignación de memoria de los WAS (ambos a 5GB recordar que se había puesto temporalmente 6GB a uno) para que queden con la misma configuración de salida a producción y el desempeño sea el mismo, y posteriormente reiniciarlos.

#### *Prueba 1 – 19:00 hs*

- 140% de carga hacia el *cluster*
- 70 usuarios virtuales adicionales directos hacia WAS1 (*.25) ejecutando TRN06
- 70 usuarios virtuales adicionales directos hacia WAS1 (*.25) ejecutando TRN07

Por problemas de configuración de la generadora de carga para esta prueba no se ejecutó la carga adicional contra el WAS2.

Los tiempos de respuesta en función del tiempo transcurrido para cada una de las transacciones que ejecutan contra el cluster se visualizan en la siguiente gráfica.




**Gráfica 27: Tiempos de respuesta en función del tiempo de ejecución**

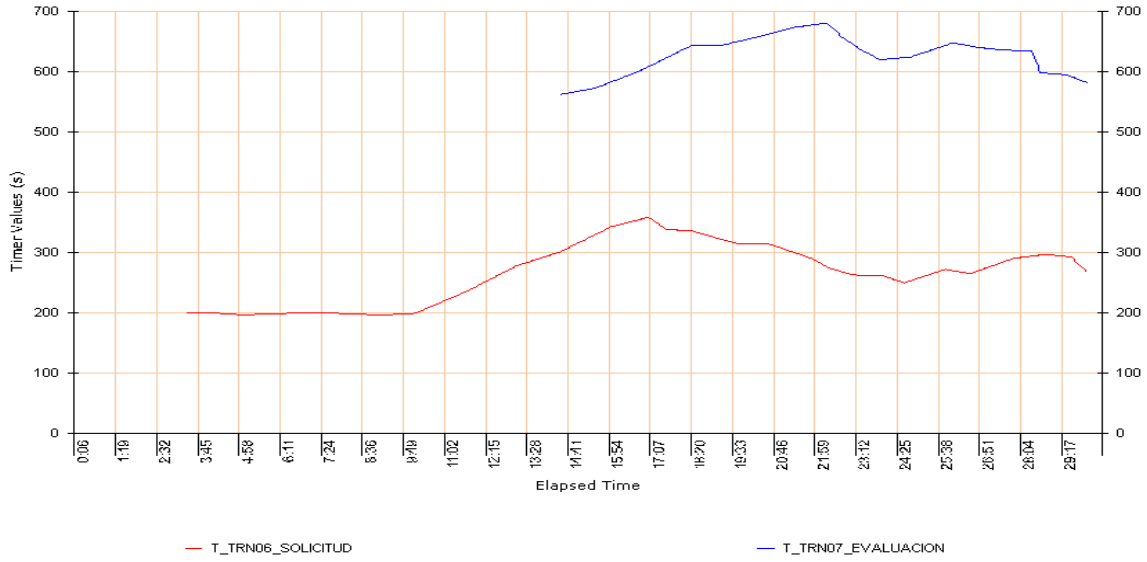
Se observa la misma oscilación en los tiempos para las TRN06 y TRN07:

- Entre 250 y 500 segundos para la TRN06
- Entre 600 y 1000 segundos para la TRN07. En este caso oscila y además se incrementan los picos con el tiempo.

Al incrementarse los tiempos para las TRN06 y TRN07 también se incrementan/desestabilizan los tiempos para las demás transacciones incluidas en la prueba, y cuando finaliza la ejecución de la TRN06 y TRN07 los demás tiempos se estabilizan. Esto muestra el alto impacto de la carga generada por las TRN06 y TRN07 en las restante transacciones.

Se detecta que al generar la carga adicional contra el WAS 25, según indica personal de DL&A, la carga en ambos WAS está equilibrada, mientras que cuando se frena la ejecución de la carga directa contra uno de los WAS la carga en ambos se desequilibra. Esto es un nuevo síntoma de que la balanceadora no está balanceando la carga correctamente.

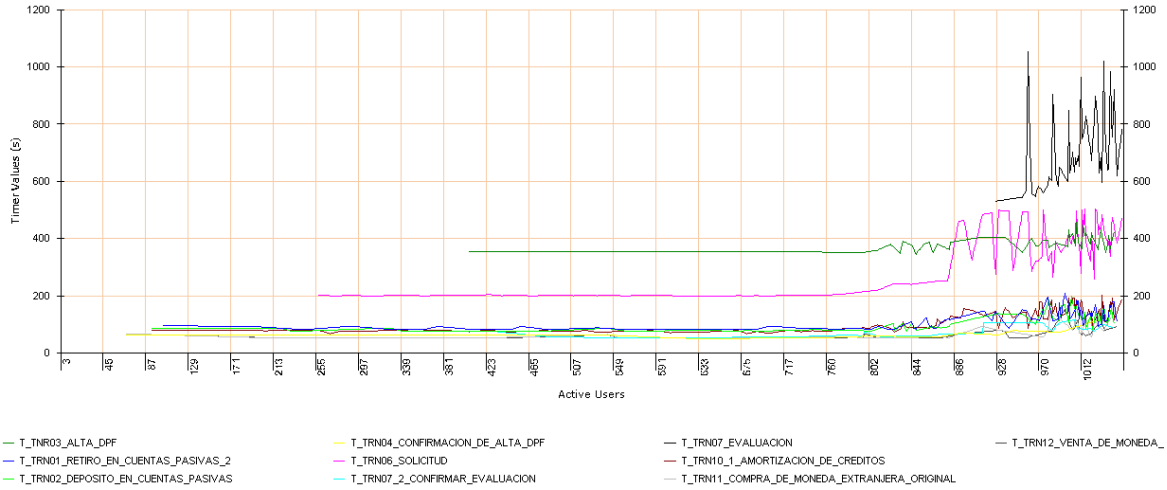
La siguiente gráfica presenta los tiempos de respuesta en función del tiempo transcurrido, obtenidos para la carga directa contra el WAS1.




**Gráfica 28: Tiempos de respuesta en función del tiempo de ejecución**

Se observa que ambas gráficas se mapean con la envolvente inferior de oscilación de los tiempos de la TRN06 y TRN07 que se ven en la gráfica presentada arriba (carga dirigida hacia el *cluster*), por lo que es de esperar que si se hubiera podido ejecutar una carga análoga directamente contra el otro WAS los tiempos obtenidos se mapearían con la envolvente superior de oscilación. Por ejemplo, para la TRN06 en ambos gráficos (teniendo en cuenta la línea inferior de oscilación) se observa que los tiempos se incrementan desde los ~10 minutos hasta los ~17 minutos de ejecución, luego descienden hasta los ~24 minutos y a partir de ahí comienzan a subir nuevamente.

La siguiente gráfica muestra los tiempos de respuesta en función de la cantidad de usuarios activos, se alcanzó un pico de usuarios activos contra el *cluster* de 1010 usuarios, a los cuales hay que sumarle aproximadamente 30 usuarios correspondientes a la carga directa contra el WAS (~1040 total).




**Gráfica 29: Tiempos de respuesta en función de la cantidad de usuarios activos**

### *Prueba 2 – 20:05 hs*

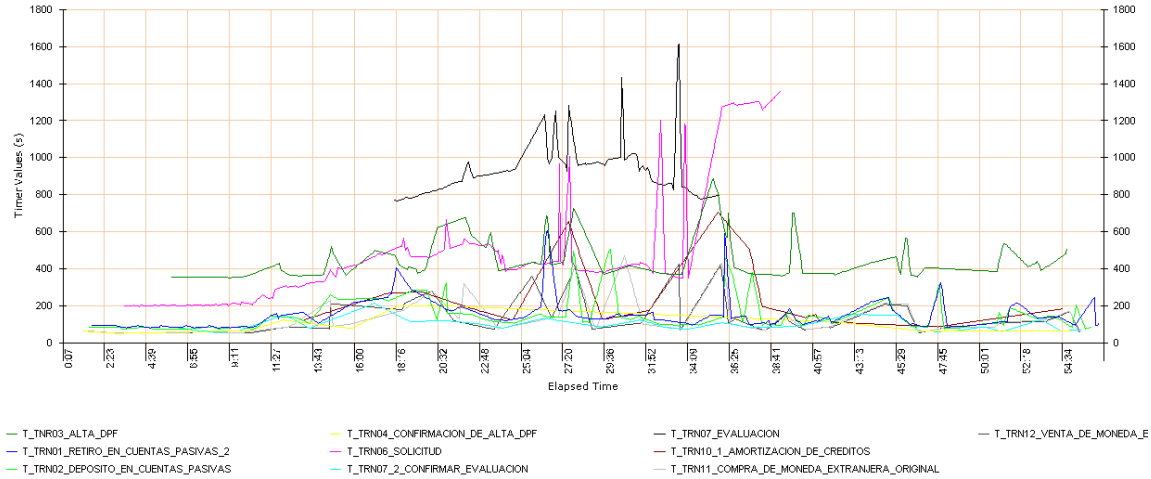
Para esta prueba personal de DL&A agrega una nueva sentencia al caché debido a un programa que estaba funcionando incorrectamente.

La carga ejecutada fue la siguiente:

- 140% de carga hacia el *cluster*
- 70 usuarios virtuales adicionales directos hacia WAS1 (*.25) ejecutando TRN06
- 70 usuarios virtuales adicionales directos hacia WAS1 (*.25) ejecutando TRN07
- 70 usuarios virtuales adicionales directos hacia WAS2 (*.25) ejecutando TRN06
- 70 usuarios virtuales adicionales directos hacia WAS2 (*.25) ejecutando TRN07

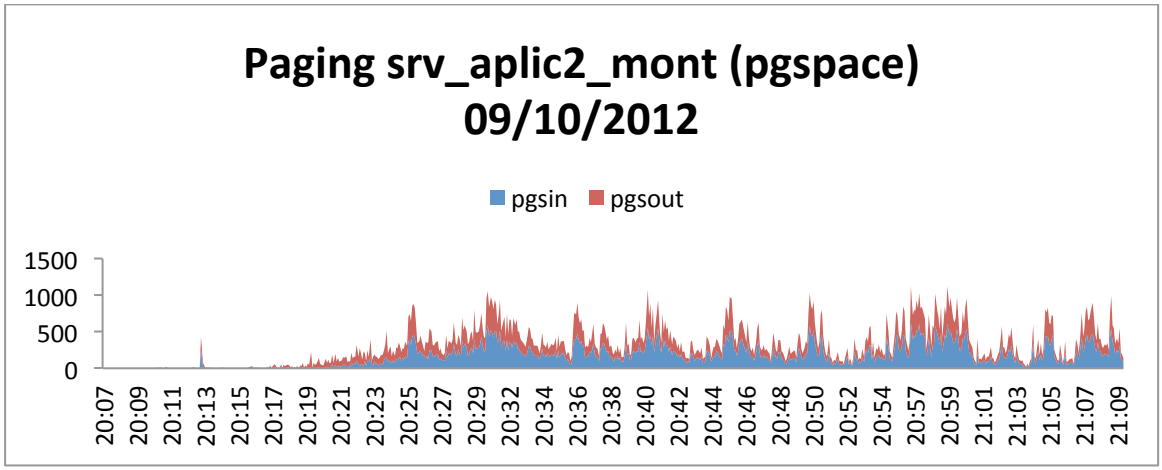
Las cargas ejecutadas directamente contra los WAS permiten obtener una carga total correspondiente al 150% de la carga esperada.

Los tiempos de respuesta en función del tiempo transcurrido para cada una de las transacciones que ejecutan contra el *cluster* se visualizan en la siguiente gráfica.




**Gráfica 30: Tiempos de respuesta en función del tiempo de ejecución**

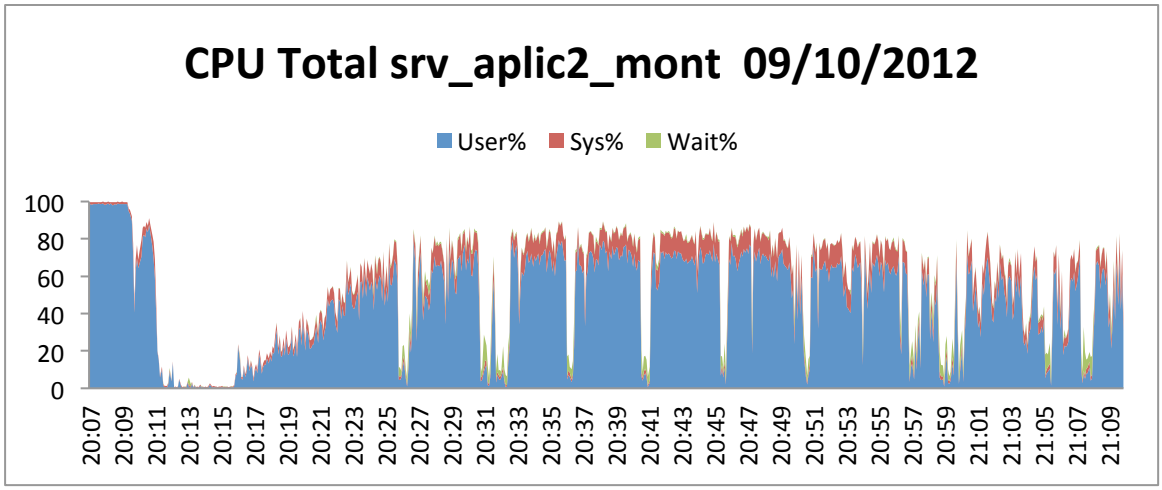
Pasada la media hora de prueba según indica el personal de DL&A los WAS comienzan a paginar demasiado. Por este motivo se recomienda revisar el modo de balanceo de la carga así como también incrementar la memoria asignada a los WAS. En las siguientes gráficas se puede apreciar el resultado de la paginación.




**Gráfica 31: Paginación - WAS (*.26)**

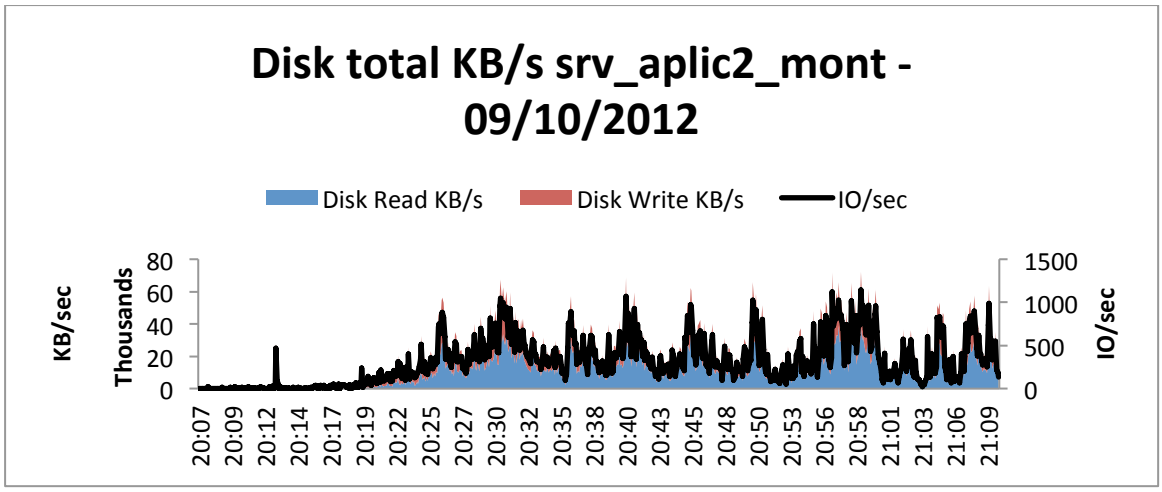
En la gráfica anterior se puede observar una serie de picos en lo que refiere al intercambio de páginas en memoria. Estos picos, como veremos a continuación se condicen con la baja de actividad en el sistema.

En la siguiente gráfica se observa como baja por momentos la actividad de la CPU. Si se observa la gráfica de CPU y la gráfica de paginación se puede ver como los picos de la primera gráfica coinciden con los de la segunda. La baja en el uso de CPU se debe a que ésta está esperando por accesos a disco dada la paginación generada.



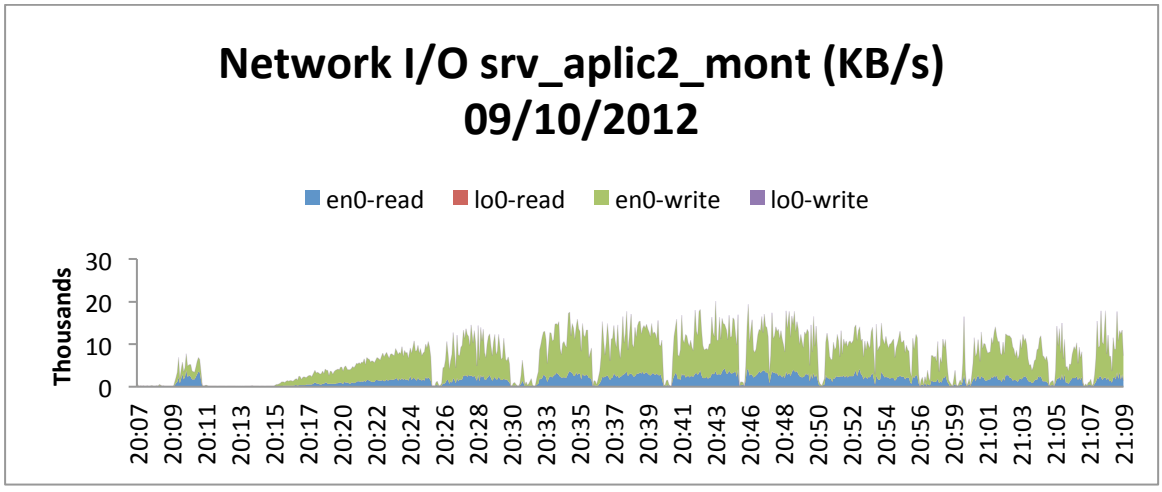

**Gráfica 32: Uso de CPU - WAS (*.26)**

En la siguiente gráfica se puede ver que los picos coinciden con la baja de CPU, dado que el CPU está esperando por datos que necesita desde el disco.



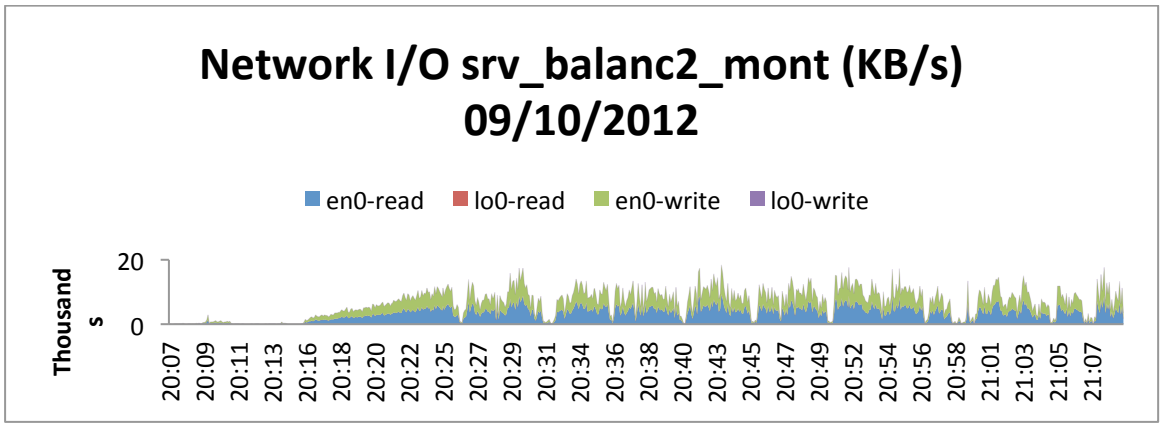

**Gráfica 33: Uso de disco - WAS (*.26)**

En la siguiente gráfica se puede ver el impacto que tiene lo anterior sobre el tráfico de red, y en consecuencia el intercambio de información ya sea peticiones o respuestas hacia el cliente (generadora de carga). Los valles en esta gráfica corresponden a momentos en los que casi no se envía o recibe información, haciendo que se demore la respuesta para el usuario.

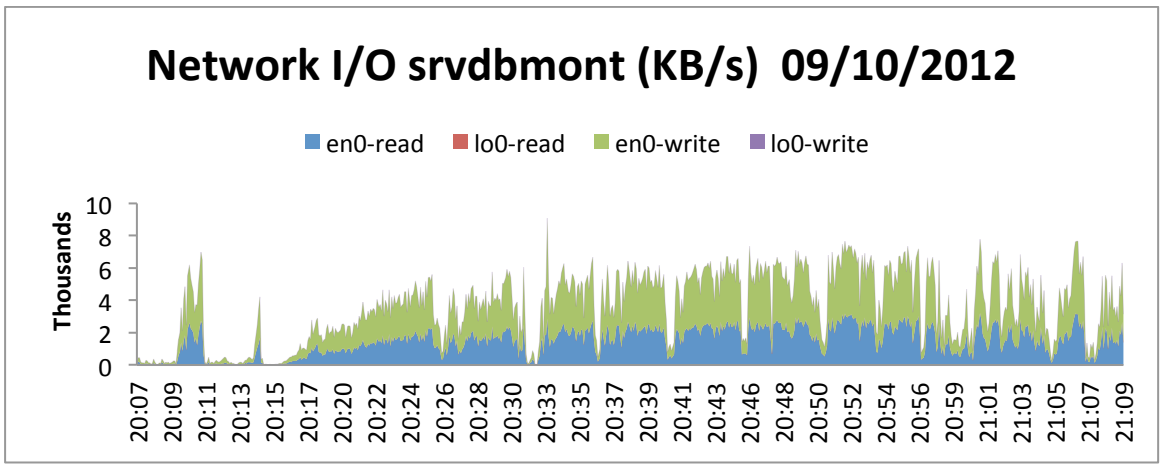



**Gráfica 34: Uso de red - WAS (*.26)**

A continuación se presenta las gráficas de red que evidencian el impacto que tiene la paginación en otros componentes del sistema como lo son la balanceadora y el servidor de base de datos.



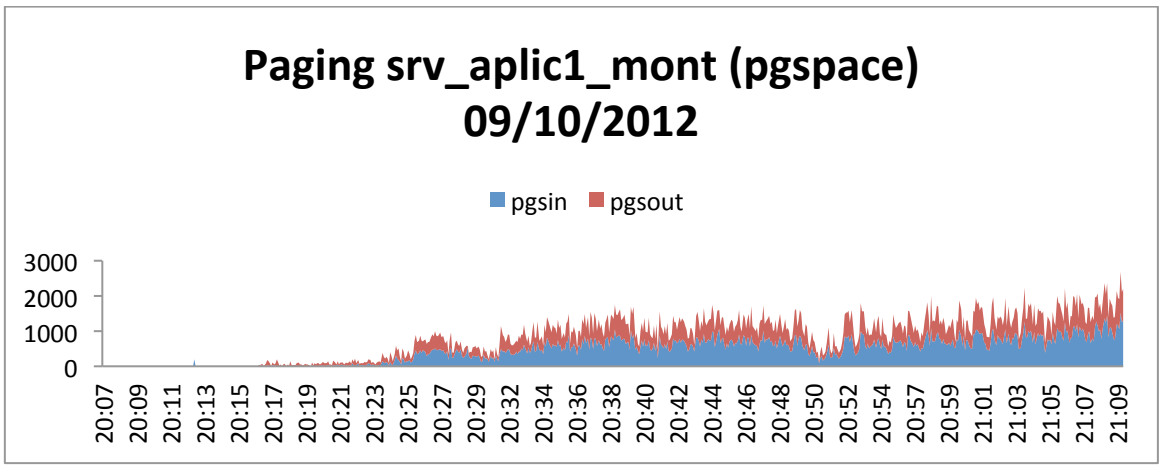

**Gráfica 35: Uso de red - Balanceadora**




**Gráfica 36: Uso de red - Base de datos**

En estos ejemplos se observa como afecta la paginación en los WAS el comportamiento del sistema. Se ve como al paginar se detienen los pedidos al servidor de base de datos y también como disminuye el tráfico hacia las generadoras.

A continuación presentamos la gráfica de paginación para el WAS1 (el cual respondió peor). Si se compara la siguiente gráfica (paginación WAS1) con la de paginación para el WAS2 se observa que la paginación en este servidor fue peor.




**Gráfica 37: Paginación WAS1**

## 3.5. Conclusiones

El proyecto lo consideramos exitoso, ya que se reprodujeron escenarios de alto uso del sistema y a la vez visualizar y solucionar varios problemas de performance previo a la puesta en producción de la versión web de Bantotal para la fusión de Caja Nuestra Gente y Financiera Confianza.

El ensayo consistió emular escenarios de carga correspondientes a la realidad de uso del sistema a través de 10 transacciones automatizadas ejecutadas por usuarios virtuales.

Se analizaron los resultados que arrojaron las ejecuciones de las pruebas sobre el SUT y se realizaron varios ajustes al sistema que mejoraron su performance y permitieron escalar la carga.

En varias pruebas los tiempos de respuesta se alejaron respecto a los *baselines*. Las transacciones asociadas al *workflow* son las que se ven más afectadas. Luego de sucesivas pruebas y evaluar cambios que conllevaron a mejoras se logró alcanzar, estabilizar y superar el 100% de la carga.

El 150% de la carga se mejoró respecto a las primeras ejecuciones, sin embargo el nivel de paginación genera que se degraden los tiempos de respuesta y no permite escalar más la carga.

Salvo por los datos, que fueron actualizados a la última migración, la infraestructura con que se salió y soportó producción fue la misma que se configuró y "tuneó" durante las pruebas.

Se recomienda ejecutar una prueba con una base de datos poblada con el volumen estimado de datos que vaya a tener la base de datos en el futuro que se desee mitigar.

Es importante aclarar que las últimas pruebas se ejecutaron en una LAN. No se consideraron en el ensayo posibles incidentes respecto a las capacidades de red en la cual se utilice el sistema. Un enlace de 4Mb fue saturado con pruebas del 10% de la carga esperada. Tal como muestran los resultados de la prueba ejecutada el día 22/08/12 una red similar no soportaría esa carga, aproximadamente 180 usuarios accediendo a la aplicación.

# 4. Central telefónica

Este capitulo describe el segundo caso es sobre el sistema de central telefónica y *contact center* de IPContact, el cual se encuentra actualmente en producción en varias organizaciones. El objetivo del proyecto fue establecer valores de configuración que permitan minimizar los riesgos asociados a cada implantación en distintas organizaciones, buscar oportunidades de mejoras en el sistema y brindar los artefactos de prueba a IPContact para futuras ejecuciones de las pruebas, con el objetivo de que quede la capacidad para seguir con esta modalidad de pruebas.

Se definieron las transacciones más importantes del negocio y automatizaron con herramientas que soportan los protocolos en cuestión, se iniciaron las ejecuciones con los *baselines*, y los escenarios se definieron ficticios basados en los resultados obtenidos. Ante la implantación en una nueva instalación, se recomienda definir los escenarios de prueba en función del uso esperado por ese nuevo sistema.

Este capitulo está estructurado de forma similar al anterior para mostrar la aplicación de la metodología y consta de una descripción de algunas de las pruebas realizadas. La primera sección, describe las pruebas desde el punto de vista de sus objetivos tanto en cuanto al software que se probó como al objetivo específico de la prueba. La sección “Escenario” describe todas las características de las pruebas desde el punto de vista del entorno de las mismas. “Automatización y armado del ambiente de pruebas” incluye aquellos temas relativos a la robotización de la prueba, punto neurálgico en la realización de una prueba de performance. En “Monitorización” se describen las herramientas utilizadas para realizar dicha actividad. “Ejecución de pruebas” describe las actividades en donde se ejecutaron los scripts antes definidos y se llevó a cabo la monitorización de la infraestructura. Finalmente, en “Conclusiones” se realizará un recuento de los resultados obtenidos y los pasos a seguir en el futuro para lograr mejoras en la performance del sistema IPContact.

## 4.1. Descripción de las pruebas

En esta sección se describe el objetivo del ensayo realizado y el software sobre el que se realizó el mismo. Estos puntos pondrán en contexto a estas pruebas y explicarán muchas de las decisiones tomadas durante la planificación del proyecto; resumen la “Etapa inicial”.

### *4.1.1. Sistema Bajo Pruebas*

El SUT es el Contact Center - IPContact versión 1.9 en conjunto con la central telefónica Asterisk versión 1.4.26. La versión probada fue desarrollada sobre una plataforma Java 1.6. La base de datos utilizada es MySQL 5.0.

Las actores que más utilizan la plataforma son los agentes, personal del *call center* que interactúa con el cliente que llama, que acceden a las principales funcionalidades a través de una aplicación web para. Esta aplicación usa un servidor web Apache 2 con un servidor de aplicaciones Tomcat 5. Cada agente, también cuenta con un terminal telefónico basado en protocolo SIP, a través de la cual se comunica por vía vocal.

Existen administradores que gestionan campañas, asignan agentes a las mismas, obtienen estadísticas del sistema, entre otras a través de otra aplicación web. También pueden obtener una monitorización del sistema en tiempo real a través de una consola.

### 4.1.2. Objetivos

Esta prueba se desarrolla con el objetivo de verificar la performance del sistema y estudiar comportamiento del mismo ante un escenario dado, representativo de la realidad en una organización. En pocas palabras, verificar que los requisitos no funcionales de performance del sistema puedan cumplirse y particularmente que los tiempos de respuesta de las transacciones sean satisfactorios para la operativa del sistema.

Este estudio implica una serie de actividades y de verificaciones a realizarse, entre las cuales se pueden enumerar:

- Tiempos de respuesta de los programas del sistema. Estos tiempos pueden ser agrupados en transacciones o sectores del ciclo funcional.
- Desempeño del servidor de bases de datos de IPContact.
- Desempeño del servidor Asterisk.

La verificación de estos parámetros, cumple la finalidad de determinar el consumo de recursos en el uso de la infraestructura, de manera que dichos datos puedan ser utilizados en el *sizing* de futuras implantaciones del sistema.

## 4.2. Escenarios

Se describe a continuación los escenarios bajo los cuales se desarrollan las pruebas que surgen del "Análisis de requerimientos". Estos escenarios se componen tanto del entorno de infraestructura que se utiliza como de los distintos escenarios de carga a reproducir con la herramienta de generación de carga.

Estas combinaciones marcan la validez de las pruebas y de los resultados a obtener en las mismas.

### 4.2.1. Escenario de Infraestructura

Para la realización de las pruebas se utilizó la infraestructura que se encuentra esquematizada en la *Figura 7*, donde también se puede observar el esquema de simulación. Nótese la existencia de dos roles diferentes en los equipos utilizados:

- **Generadores de carga (SIPp y JMeter)**: son los responsables de generar la carga (pedidos SIP y HTTP) de cada uno de los usuarios virtuales que participan en la prueba. Se utilizó una máquina para generar el tráfico correspondiente a las llamadas telefónicas y otra para el tráfico correspondiente a la aplicación web de agentes, aplicación web de administrador (consultas web) y la consola.
- **Servidores:** responsables de realizar el procesamiento de los pedidos de los usuarios virtuales que se incluyen en la prueba. Cada uno consiste en una máquina física con Sistema Operativo CentOS 5.3 y cumplen diferentes roles:
    - Testing 1 (IP: 192.168.3.150)
        - **Servidor de aplicación (Java):** se comunica con el Servidor Asterisk y proporciona funcionalidades de más alto nivel como *login* de agentes y generación de llamadas.
        - **Servidor web:** permite gestionar y operar las actividades del *call center*.
        - **Servidor AGI:** permite ejecutar código Java desde Asterisk. Usado básicamente para realizar operaciones de acceso a base de datos.
        - **Servidor de base de datos (MySQL):** DBMS con los datos del sistema.

- Testing 2 (IP: 192.168.3.151)
  - **Servidor Asterisk:** proporciona funcionalidades de una central telefónica.
  - **IVR:** brinda un menú vocal a quien llama, a través del teclado del teléfono se pueda escoger una opción que representa la respuesta o servicio buscado. Está programado dentro de Asterisk, no es otro componente.

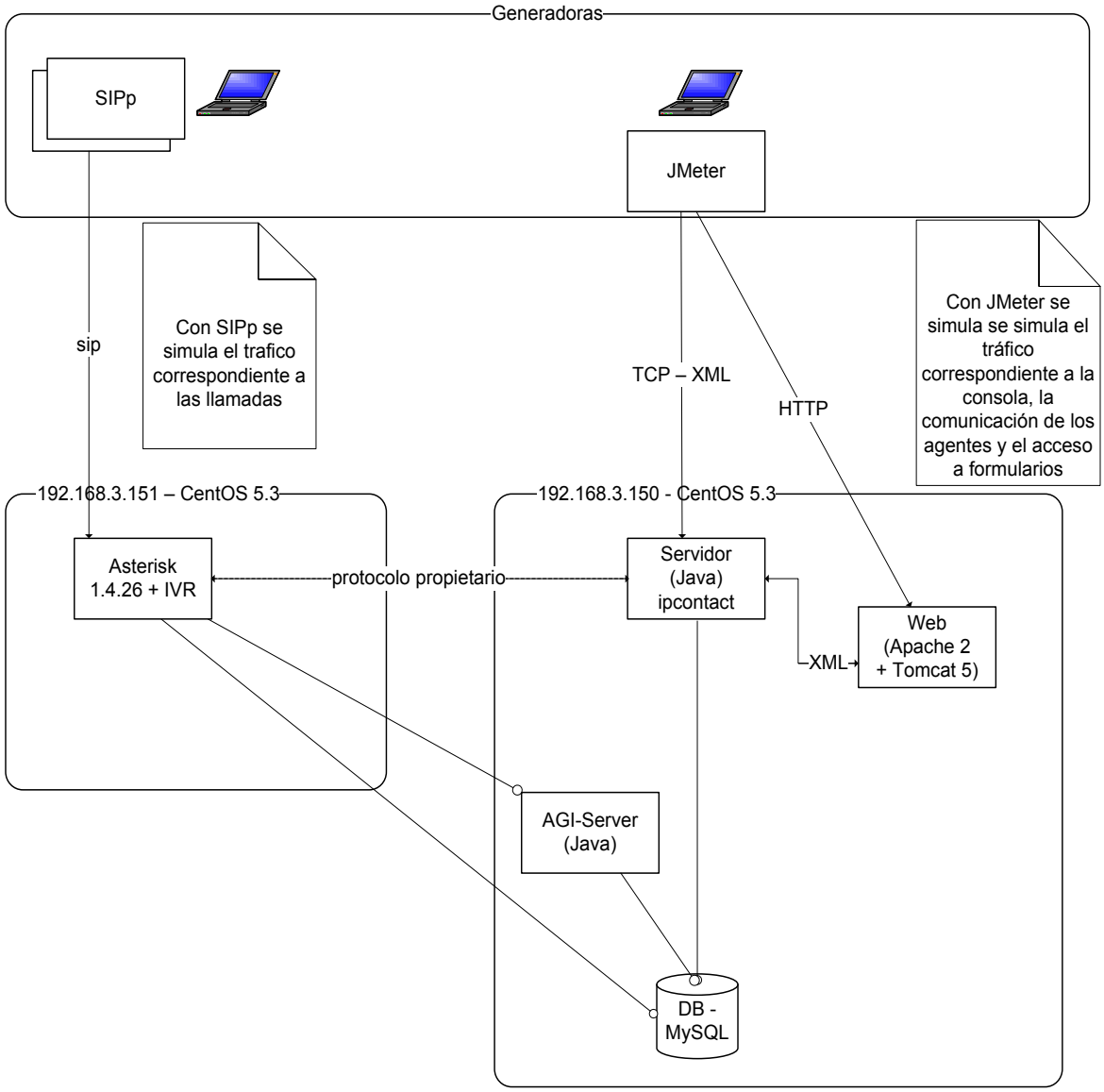



**Figura 7 - Esquema de la infraestructura de pruebas**

### *4.2.2.Escenario de Carga*

Sobre el escenario de infraestructura establecido, se ejecutaron escenarios de carga compuestos por:

- Transacciones que participan en la prueba
- Usuarios del sistema.
- Cantidad de usuarios que ejecutan cada una de las transacciones.
- Cantidad de transacciones que ejecutan en una hora (*Throughput*) para cada transacción.
- Tiempos en los que los usuarios del sistema no interaccionan (*Think times* y *Delays*)

Este escenario fue definido en conjunto con IPContact, basado en un estudio del uso típico del sistema hoy en día implantado en varios clientes y con el objetivo de que pueda ser usado para facilitar las proyecciones del uso del mismo en diferentes contextos.

#### 4.2.2.1. Transacciones seleccionadas

Se seleccionaron un total de ocho transacciones para la simulación, tres transacciones de la operativa del sistema web y cinco que involucran el uso de la central Asterisk. A continuación se describen las transacciones seleccionadas.

**TRN01_Campaña de Salida**: aquí el agente recibe un identificador de campaña y solicita los datos de un cliente en ésta campaña al cual llamar. Una vez recibido los datos del cliente, el agente da la orden de iniciar la llamada.

**TRN02_Campaña Automática**: en este caso es la central quien realiza llamadas a los clientes involucrados en dicha campaña y una vez que este atiende, es comunicado con el agente.

**TRN03_Agente** no está en campaña: refiere la recepción de llamadas de clientes, desde el exterior de la empresa, que serán despachadas por la cola de llamadas y asignadas a los agentes disponibles en ese momento.

**TRN04_IVR**: emula la acción de un cliente que llama desde el exterior de la empresa y es atendido por un IVR. Este solicita que ingrese la cédula, y en el caso que la cédula sea correcta finaliza la transacción, en otro caso, la llamada es derivada a un agente disponible en ese momento.

**TRN05_Administrador**: refiere a un usuario que se autentica como administrador en una consola y a partir de ahí, comienza a recibir reportes y datos del tráfico existente en el sistema a lo largo del tiempo.

**TRN06_Datos de Colas**, **TRN07_Actividad de Agentes** y **TRN08_Registro de llamadas** son transacciones web, en las cuales un administrador/supervisor ingresa al sistema y emite reportes de estas actividades en un rango de tiempo indicado.

A continuación en la *Figura 8*, se presenta un diagrama del funcionamiento de las transacciones que están involucradas con el uso de la central Asterisk.

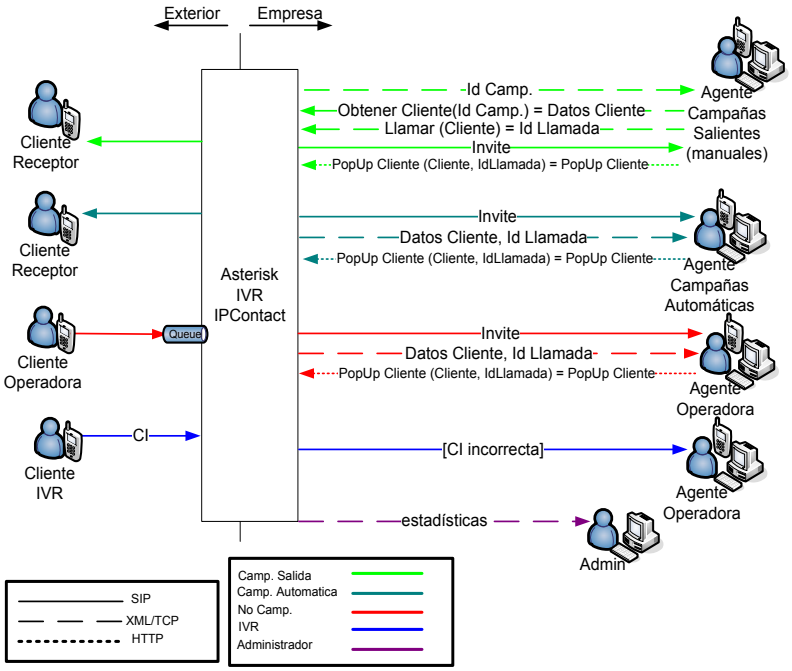



**Figura 8 - Esquema de funcionamiento de transacciones**

#### 4.2.2.2. Métricas de las transacciones

Luego de analizar el funcionamiento de las transacciones, se establecieron algunas métricas para los tiempos de respuesta. Para esto en cada transacción se establecieron los pasos en los cuales es crítico obtener una rápida respuesta del sistema, con el fin de focalizar el análisis sobre estos.

Se detalla a continuación para cada transacción cuáles fueron los puntos que se consideraron más importantes y cuál fue el motivo de la elección de los mismos.

### TR01 Campaña de salida

En esta transacción la métrica considerada importante es la diferencia de tiempo desde que se origina la llamada (1) hasta que la llamada llega al Agente (2). También es importante que una vez que el Agente recibe la llamada, tenga el popup3 (3) correspondiente a la llamada con los datos del cliente. En la *Figura 9* se marcan en azul la secuencia y la acción que determina cada uno de estos eventos.

Para el análisis de esta transacción se consideran las siguientes definiciones de medidas de tiempos:

- Δ ring = SIP Invite (2) – Originate (1)
- Δ dataCallReady = Respuesta Popup (3) – SIPInvite (2)
- Δ dataCallOriginate = Respuesta Popup (3) – Respuesta Originate (1)

[3] Ventana que recibe el Agente con datos del usuario que se está llamando.

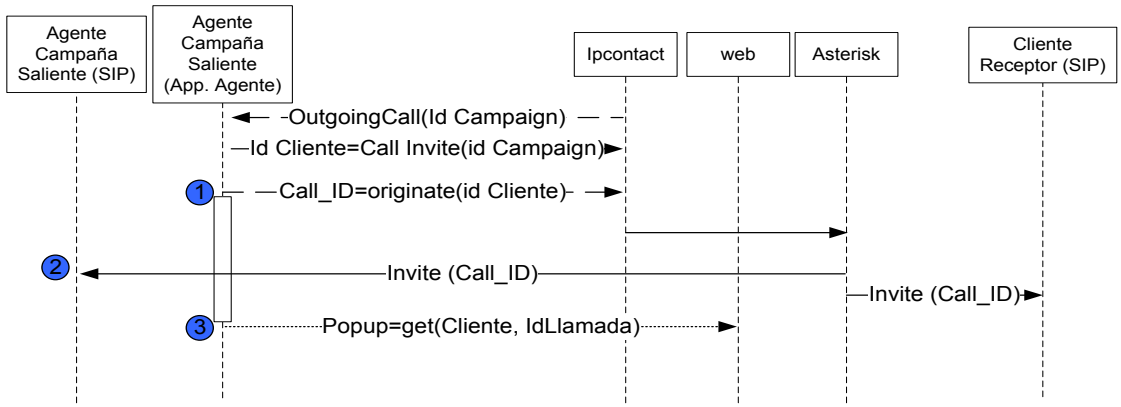



**Figura 9 - Métricas TRN01**

## TR02 Campaña Automática

En esta transacción se considera importante medir la diferencia de tiempos entre que la llamada llega al agente (1) y ésta es registrada en la central (2). También que cuando el agente está en condiciones de hablar, el Popup con los datos de clientes ya esté disponible en su pantalla (3). Interesa por otro lado que el tiempo entre que la llamada es registrada en la central (2) y el Popup llega al Agente (3) se mantenga, a lo largo de la prueba (y ante incrementos de la carga), constante. Los eventos mencionados se esquematizan en la *Figura 10*.

Para el análisis de esta transacción se consideran las siguientes definiciones de medidas de tiempos:

- Δring = Call Campaign (2) - SIP Invite (1)
- Δ dataCallReady = Popup Respuesta (3) – SIP Invite (1)
- Δ dataCallCampaign = Popup Respuesta (3) – Call Campaign Respuesta (2)

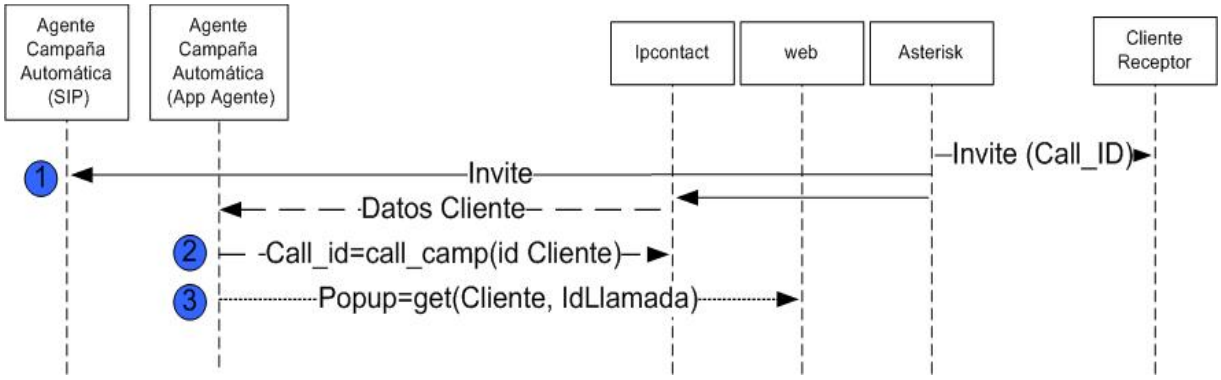



**Figura 10 - Métricas TRN02**

## TRN03 Agente no está en campaña

La única diferencia entre esta transacción y la anterior es que aquí quien genera la llamada es un cliente externo y no la central. Por lo tanto los tiempos a considerar son:

- Δring = Call Campaign (2) - SIP Invite (1)
- Δ dataCallReady = Popup Respuesta (3) – SIP Invite (1)
- Δ dataCallCampaign = Popup Respuesta (3) – Call Campaign Respuesta (2)

Estos eventos se marcan en azul en la *Figura 11*.

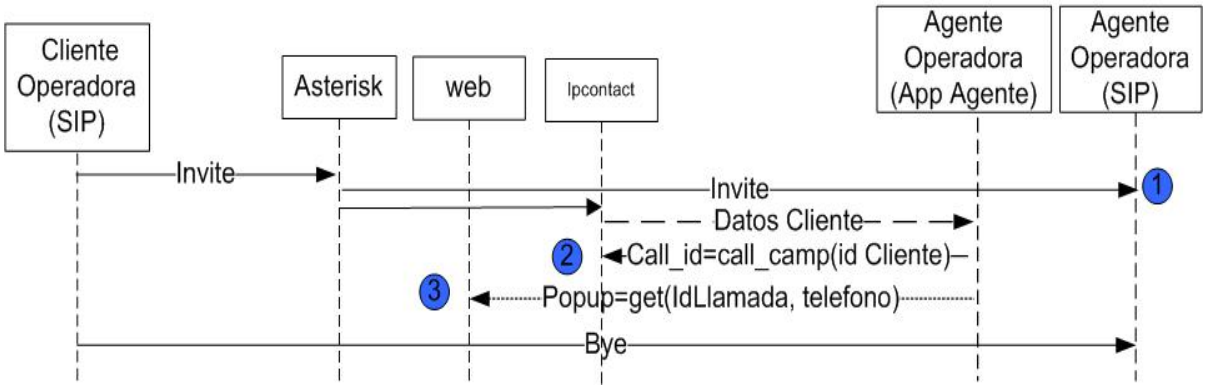



**Figura 11 - Métricas TRN03**

### TRN04 IVR

Para esta transacción, como no se recibe *popup* con datos del cliente y la duración de la llamada es constante (generada por el cliente automatizado), no se utilizaron métricas de tiempo Asterisk-IPContact, sino que se controló la integridad de la llamada, y que se esté despachando el total de las mismas. En la *Figura 12* se puede ver cómo interactúa esta transacción.

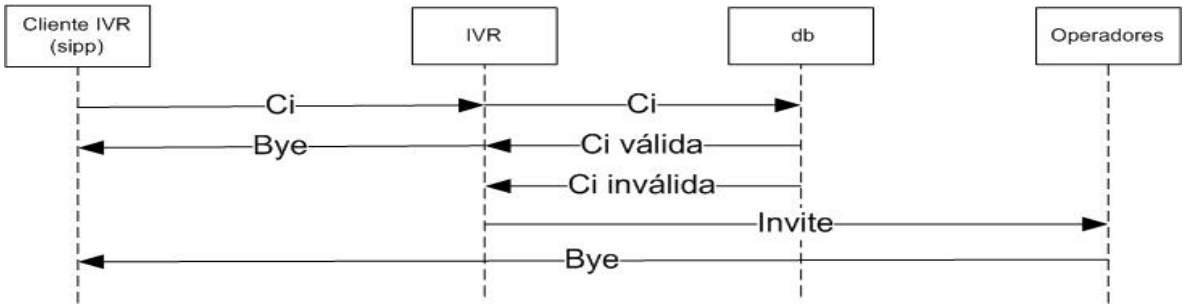



**Figura 12 - Funcionamiento TRN04**

#### 4.2.2.3. Mezcla de transacciones

Para definir el escenario en el cual se iba a correr las pruebas, se comenzó ejecutando los *baselines* para observar el comportamiento de cada transacción en el sistema y la infraestructura planteada. Al observar que las transacciones que involucran al Asterisk respondían en tiempos aceptables y que las transacciones web consumían todo el CPU del servidor en el que corre la base de datos, se decide partir de un escenario de carga inicial que solamente involucre un usuario virtual de cada transacción web. Este escenario se puede observar en la siguiente tabla.

Partiendo de esta carga inicial y observando las respuestas del sistema luego de cada ejecución, se fueron generando distintas mezclas de transacciones con el objetivo de probar el sistema desde distintos enfoques.

| Transacción | Usuarios concurrentes | **Iteraciones** (Por hora Por usuario) | Total de datos a procesar |
|---|---|---|---|
| TRN01_Campaña de Salida | **5** | **514** | 2570 |
| TRN02_Campaña Automaticas | **5** | **600** | 3000 |
| TRN03_Agente no está en Campaña | **5** | **600** | 3000 |
| TRN04_IVR | **5** | **327** | 1635 |
| TRN05_Administrador | **5** | **1** | 5 |
| TRN06_Datos de Colas | **1** | **13** | 13 |
| TRN07_Actividades de Agentes | **1** | **17** | 17 |
| TRN08_Registro de llamadas | **1** | **19** | 19 |
| **Total** | **28** | | |

### *4.2.3.Datos utilizados*

Para la realización de las pruebas se utilizaron datos extraídos de una base que actualmente está en producción complementada con los datos necesarios para correr las pruebas.

El objetivo de la generación de datos fue representar el volumen de datos con el que generalmente trabaja el sistema en producción.

Los datos que se agregaron son los correspondientes a las "Campañas Automáticas", "Campañas de Salida", los "Agentes" y "Administradores" que se usaron en la simulación.

Los volúmenes de datos involucrados son los siguientes:

| Nombre | Registros | MB |
|---|---|---|
| callinfo | 738790 | 34,09 |
| cdr | 731683 | 0,05 |
| callmonitor | 346905 | 0,03 |
| agent_lost | 242242 | 131,91 |
| users_log | 42058 | 72,34 |
| log | 15632 | 0,02 |
| links | 6961 | 219,02 |
| clientes_extrainfo | 2153 | 0,02 |
| clientes | 2066 | 0,02 |
| datos_cedula | 1054 | 0,02 |
| queue_users | 840 | 0,06 |
| agents | 335 | 0,03 |
| users | 308 | 0,03 |
| queue | 139 | 0,03 |
| authcalls | 50 | 0,03 |
| extension | 41 | 0,02 |
| sip_account | 41 | 0,06 |
| llamadas_resultados | 14 | 0,03 |

## 4.3. Automatización y armado del ambiente de pruebas

### *4.3.1. Herramientas*

Se utilizaron dos herramientas para generar el tráfico existente en el sistema. La herramienta seleccionada para automatizar las transacciones que utilizan el protocolo SIP, correspondiente a las llamadas telefónicas tanto sea de agentes como clientes, ha sido SIPp. Para generar el tráfico HTTP y TCP correspondiente a la interacción de los agentes con el sistema IPContact se utilizó JMeter. De está manera se emulan las llamadas SIP y el tráfico HTTP correspondiente a la solicitud del *popups* con los datos de quien llama y las consultas web hechas por los administradores, como también el tráfico TCP correspondiente al diálogo mediante XML que mantienen tanto la Consola como los Agentes. Al tratase de herramientas libres que se encuentra bajo licenciamiento GPL su utilización no tuvo costo adicional para el proyecto.

Algunas de las decisiones que llevaron a utilizar éstas herramientas para el proyecto fueron:

- Soporte del tráfico necesario para la integración de ambos sistemas
    - Telefonía e informática
- Sin costo de licenciamiento.
- Facilidad para generar logs.
- Posibilidad de escalar la carga rápidamente.
- Posibilidad de emular escenarios de concurrencia.

Para reproducir las transacciones que interactúan con Asterisk, las herramientas SIPp y JMeter trabajan coordinadas, de manera que entre ambas puedan simular el comportamiento real de un usuario. De esta forma mientras JMeter simula la actividad que los usuarios realizan contra el sistema IPContact (ej. registro, recepción de datos para llamadas), SIPp se utiliza para simular el tráfico de llamadas que realizan estos agentes.

### *4.3.2. Monitorización*

Durante el despliegue del "Escenario de Infraestructura" definido para la simulación (es decir el "Armado del ambiente de pruebas") también se configuraron los monitores de primer nivel que recolectaron datos para detectar posibles cuellos de botella.

### *4.3.3. Indicadores*

Los siguientes indicadores fueron recolectados y estudiados luego de cada prueba. En este documento se presentan algunos elementos del monitoreo que facilita la comprensión de cada situación.

- **Servidor de base de datos e IPContact**: Uso de CPU total y dedicada a base de datos, paginado de usuario, uso físico del disco, CPU de disco, uso de LAN.
- **Servidor de aplicación**: Uso de memoria (Heap de JVM), Tiempo de Garbage Collection.
- **Servidor Asterisk**: Uso de CPU total, uso físico del disco, CPU de disco, Memoria disponible, uso de LAN.

### *4.3.4. Herramientas utilizadas*

Se utilizan varias herramientas para realizar la tarea de monitorización.

- **Nmon:** permitió la monitorización de los sistemas Linux (generadora de carga, servidor Asterix, servidor con IPContact y base de datos).
- **GCViewer:** permitió la interpretación gráfica de los *logs* de *Garbage Collector*.

## 4.4. Ejecución de pruebas

En las primeras pruebas se detectaron problemas con las consultas web, esto motivo variar los escenarios definidos inicialmente, se incluyeron escenarios que no ejecutaban consultas web, otros que ejecutaban una de cada tipo y otros que ejecutaban todas las consultas. El objetivo de este cambio fue observar el impacto que tiene en la performance global del sistema cada una de las transacciones web.

La finalidad de los distintos escenarios fueron las siguientes:

- Estudiar el comportamiento del sistema ante las distintas cargas del sistema.
- Ajustar la infraestructura y el ambiente a medida que se incrementa la carga.

En esta sección se describen las pruebas ejecutadas y los resultados encontrados en las mismas.

### *4.4.1. Cronograma de las pruebas*

Las pruebas se ejecutaron entre el 7 de abril y el 25 de mayo de 2010. La *Figura 13* muestra la distribución en el tiempo de las ejecuciones de las pruebas más relevantes.

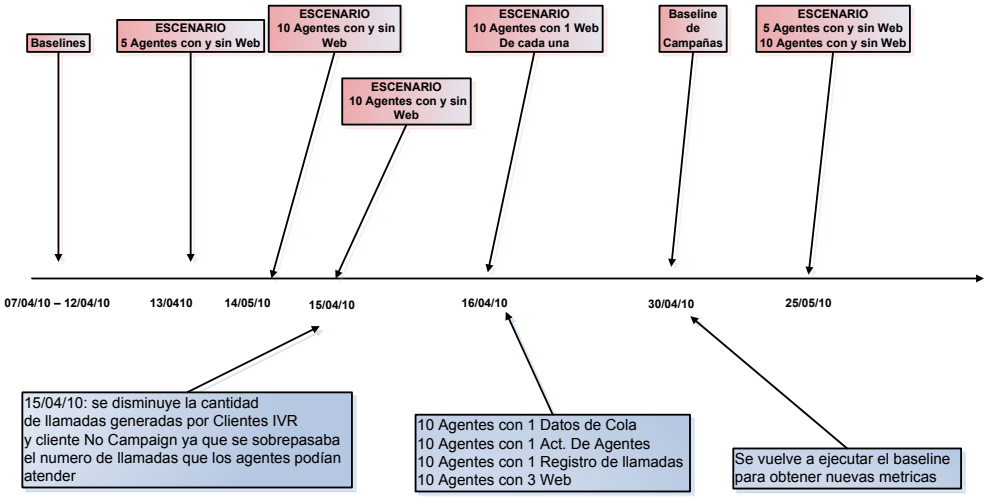



**Figura 13 - Calendario de las principales ejecuciones y cambios**

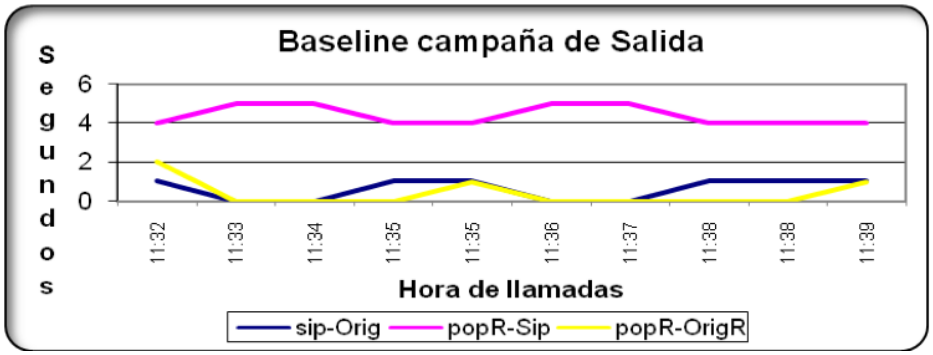
### *4.4.2. Resultados*

A continuación, se presentarán los resultados más relevantes de las pruebas ejecutadas sobre el sistema. No todas las ejecuciones son descriptas en esta sección, solo aquellas que arrojaron resultados de relevancia o que permitieron verificar el comportamiento del sistema ante un cambio sobre la infraestructura o sobre la lógica de la aplicación.

#### 4.4.2.1. Tiempos base

Esta prueba consistió en la ejecución de 10 iteraciones de cada una de las transacciones. Se busca obtener indicadores estadísticos de los tiempos de respuesta óptimos. Para esto se hizo un análisis de las medidas detalladas en la sección “Métricas de las transacciones”.

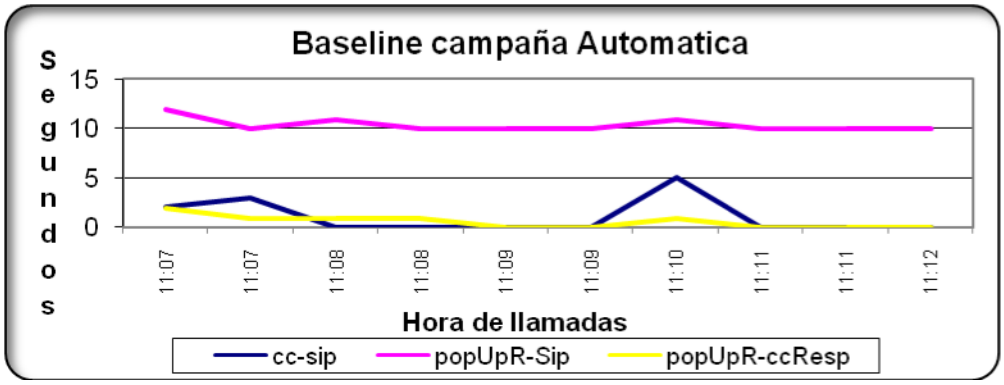
La siguiente gráfica presenta estas métricas para el *baseline* de “Campaña saliente”. De la misma se desprende que la diferencia de tiempo entre que se origina la llamada y la misma llega al Agente varía entre cero y un segundo. La diferencia entre que la llamada llega al agente y los datos del cliente (al cual se llama) son presentados en el Navegador del Agente, varía entre cuatro y cinco segundos, mientras que la diferencia entre la respuesta del origen de la llamada (*originate*) y la respuesta del *popup* varía de cero a un segundo.




**Gráfica 38: *Baseline* "Campaña saliente"**

Durante la ejecución de este *baseline* el CPU de la máquina testing1 presento un pico del 20% de uso, el resto del tiempo no supero el 5%. La máquina testing2, presenta actividad menor al 1%.

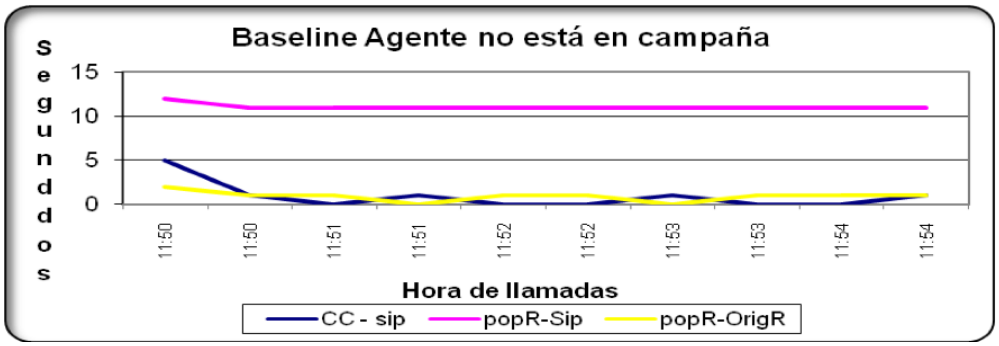
En la siguiente gráfica se muestran los tiempos de respuesta del *baseline* de la "Campaña automática". Se observó que la diferencia de tiempo entre que la invitación de la llamada llega al agente y se produce el evento *Call Campaign* varía entre cero y cuatro segundos. La diferencia de tiempo entre que la llamada llega al Agente y los datos del cliente llamado son mostrados en el monitor del Agente varía en un entorno de 10 segundos. Por otro lado, la diferencia de tiempo entre la respuesta del pedido *Call Campaign* y la respuesta del *popup* varía de cero a un segundo.




**Gráfica 39: *Baseline* "Campaña automática"**

Las máquinas testing1 y testing2 prácticamente no mostraron actividad durante la ejecución.

A continuación en la siguiente gráfica se muestran los tiempos obtenidos en el *baseline* de la transacción "Agente no está en campaña". En este caso se observó que la diferencia de tiempo entre que la invitación de la llamada llega al agente y se produce el evento *Call Campaign* varía entre cero y un segundo, el valor de cinco segundos en la primer muestra podría descartarse ("tiempo de calentamiento"). La diferencia entre que la llamada llega al agente y los datos del cliente llamado son mostrados en el monitor del Agente está en un entorno de los 11 segundos. Mientras que la diferencia entre la respuesta del pedido *Call Campaign* y la respuesta del popup varía en un entorno de cero a dos segundos.



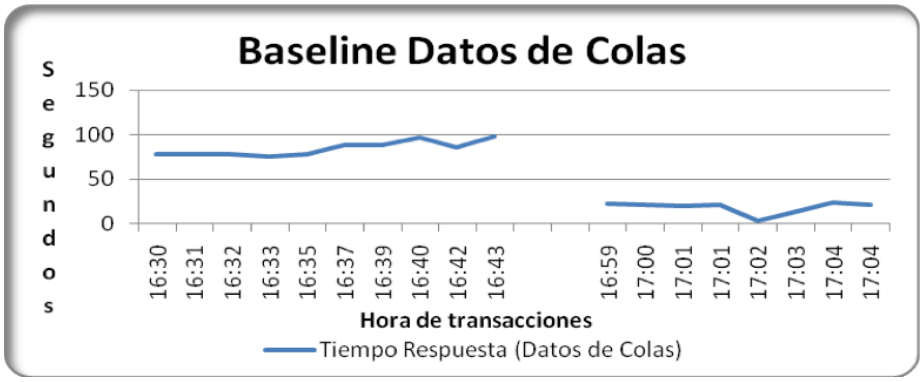

**Gráfica 40: *Baseline* "Agente no está en campaña"**

Al igual que en los *baselines* anteriores, las máquinas que alojaban el SUT estuvieron holgadas.

El *baseline* de la "IVR" fue inmediato y correcto. En el caso del "Administrador", el *baseline* consiste en autenticarse al sistema, no generan datos ya que es la única transacción ejecutando.

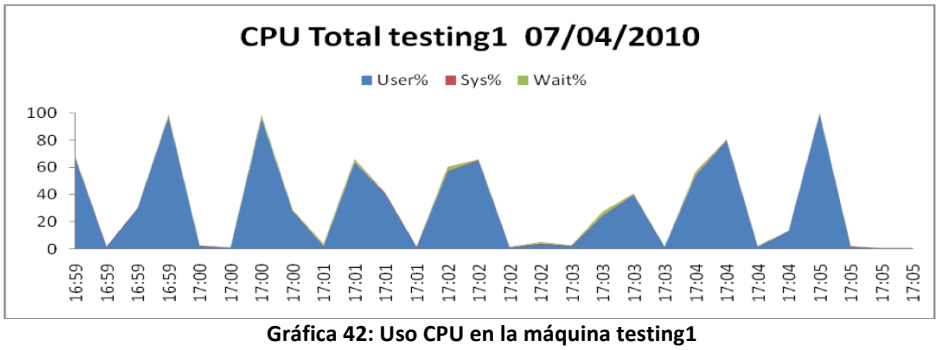
Para la transacción "Datos de Colas" se toman dos ejecuciones del *baseline*. Una seguida de la otra con los mismos datos para comprobar el comportamiento del DBMS luego de que los datos son guardados en memoria cache.

Como se muestra en la siguiente gráfica los tiempos obtenidos en la primera ejecución son muy diferentes a los tiempos obtenidos en la siguiente ejecución donde los datos ya se encuentran en memoria cache. Para la primera ejecución los tiempos fueron del orden de 80 a 100 segundos, mientras que la segunda ejecución los tiempos se mantuvieron inferiores a los 20 segundos.



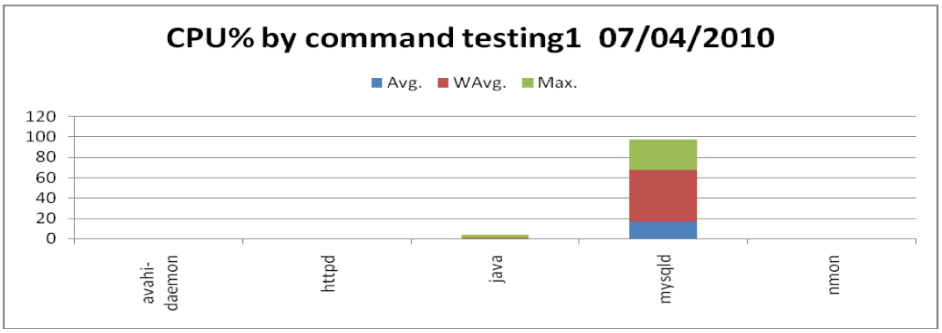

**Gráfica 41: *Baselines* "Datos de colas"**

En la siguiente gráfica se muestra el uso del CPU durante la segunda ejecución. Se puede observar que aún cuando los datos están en memoria cache, y los tiempos mejoran en gran medida, el procesador alcanza picos del 100%.



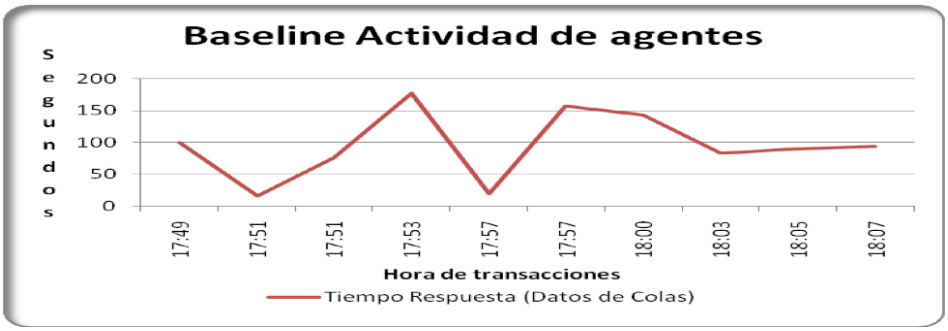

**Gráfica 42: Uso CPU en la máquina testing1**

Se comprobó que la actividad que satura el CPU es la asociado al motor de base de datos.



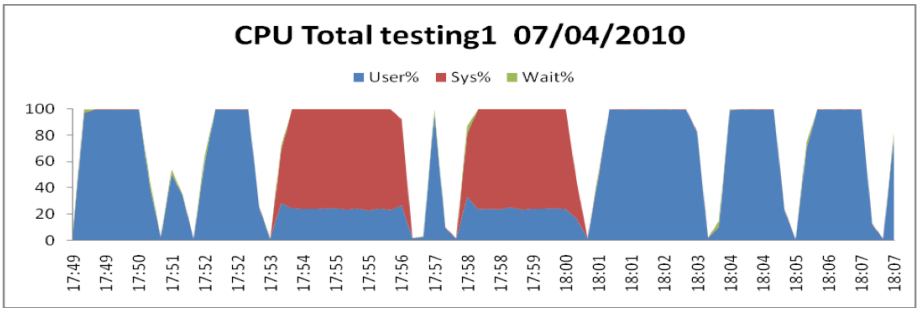

**Gráfica 43: Procesos baseline Datos de colas**

En la transacción "Actividad de agentes" se observó el mismo comportamiento en cuanto el uso de memoria *cache* para los datos. Se puede observar en la Gráfica 44 que los tiempos en distintas ejecuciones de la transacción varían desde 10 a 150 segundos, dependiendo si el juego de datos utilizado se encuentra en memoria *cache* o no.



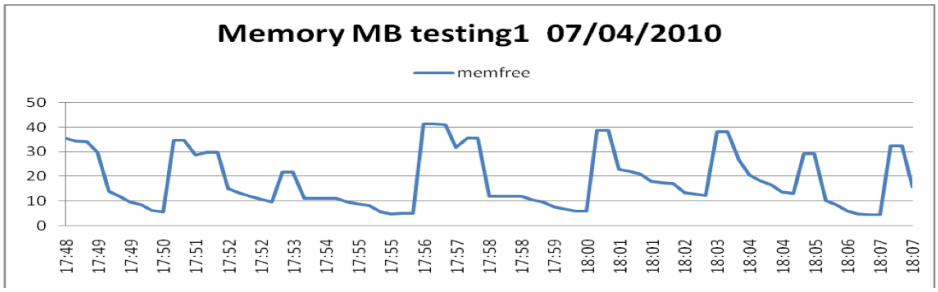

**Gráfica 44 - *Baseline* "Actividad de agentes"**

Se analizó el uso del CPU durante esta transacción y como muestra la siguiente gráfica se presentaron picos sostenidos en un entorno del 100% durante casi toda la prueba. Se observó que en el caso de ejecutar con datos que se encuentran en el cache de la máquina, el pico de CPU es inferior y permanece un período menor de tiempo.




**Gráfica 45 - Uso CPU en la máquina testing1**

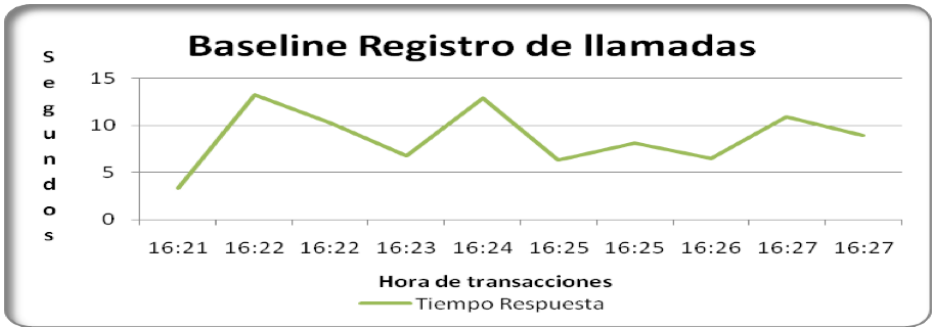
Como se muestra en la siguiente gráfica, el uso del CPU se debe a que la máquina se queda sin memoria con lo cual es inevitable el paginado y esto repercute en ciclos del CPU.




**Gráfica 46 - Uso de memoria en la máquina testing1**

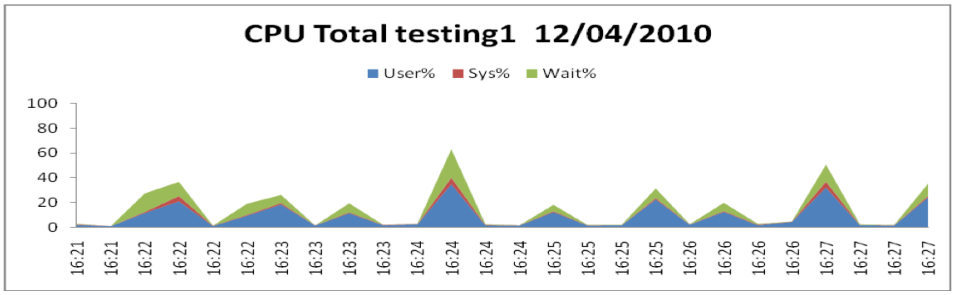
Se verificó que, nuevamente, es MySql quién desencadena el comportamiento mencionado.

En la transacción "Registro de llamadas" se observó que los tiempos de respuesta fueron mucho menores a los obtenidos hasta el momento por las transacciones web. Esto se puede ver en la siguiente gráfica. Los tiempos en este caso varían en torno a los 10 segundos.




**Gráfica 47: *Baseline* "Registro de llamadas"**

Al observar cómo afecta al CPU esta transacción, se puede ver que éste se mantiene holgado obteniendo picos como máximo de un 40% en los peores casos. El resto de las veces despacha la transacción con un esfuerzo de entre un 10 y 15%. Esto se muestra en siguiente gráfica.




**Gráfica 48: Uso del CPU *baseline* "Registro de llamadas"**

Del análisis de las transacciones web, se ve las transacciones "Datos de colas" y "Actividades de agentes" saturan el procesador de la base de datos debido a la paginación, esta se desencadena por actividad del DBMS (MySql), es importante destacar que el equipo tiene antes de la prueba 30MB libres.

A raíz de estas pruebas, se concluye que no amerita escalar en cuanto a los usuarios virtuales ejecutando transacciones web, ya que se alcanzó la saturación de un recurso con esta carga. Este resultado es importante al momento de armar los siguientes escenarios de prueba. Para seguir escalando en este sentido sería necesario superar los problemas detectados en esta prueba.

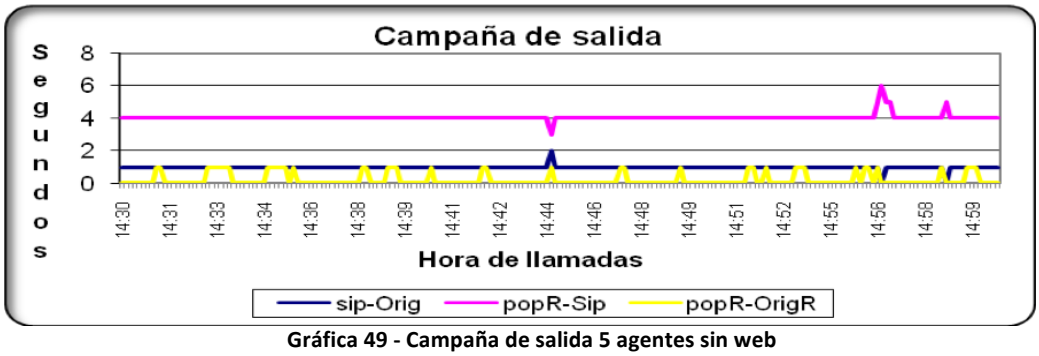
### 4.4.2.2. Prueba 5 Agentes sin Web

A raíz del resultado obtenido en los *baselines*, se decide probar con un escenario donde no se incluyan consultas web. El objetivo es probar la carga que producen las campañas, teniendo la máquina dedicada solamente a estas transacciones. El escenario se puede observar en la siguiente tabla.

| | Cinco agentes sin web | | | |
|---|---|---|---|---|
| **Transacción** | **Agentes/ usuarios virtuales** | **Duración de la llamada/ transacción (en segundos)** | ***Delay*** **entre iteración** | **Cantidad de llamadas / iteraciones en una hora** |
| **TRN01_Campaña de Salida** | **5** | **30** | **5** | **514** |
| **TRN02_Campaña Automáticas** | **5** | **30** | **0** | **600** |
| **TRN03_Agente no Está en Campaña** | **5** | **30** | **0** | **600** |
| **TRN04_IVR** | **5** | **50** | **5** | **327** |
| **TRN05_Administrador** | **5** | **N/A** | **N/A** | **N/A** |
| **TRN06_Datos de Colas** | **0** | **N/A** | **N/A** | **N/A** |
| **TRN07_Actividades de Agentes** | **0** | **N/A** | **N/A** | **N/A** |
| **TRN08_Registro de llamadas** | **0** | **N/A** | **N/A** | **N/A** |

**Tabla 3 - Escenario 5 agentes sin web**

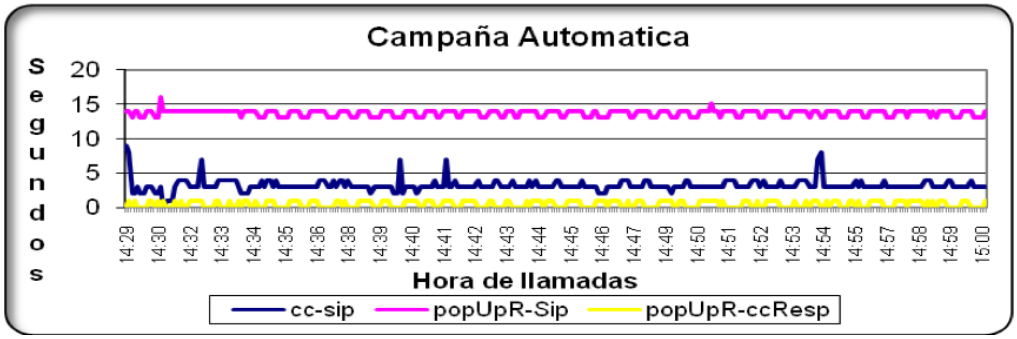
Comparando los resultados obtenidos en esta prueba para la transacción "Campaña de Salida" (que se muestran en la Gráfica 49), con el *baseline* de esta misma se observa que los tiempos de respuesta se mantienen.




**Gráfica 49 - Campaña de salida 5 agentes sin web**

La siguiente gráfica muestra los tiempos de respuesta de la "Campaña Automática". Se puede observar un incremento de unos pocos segundos respecto a los *baselines*.

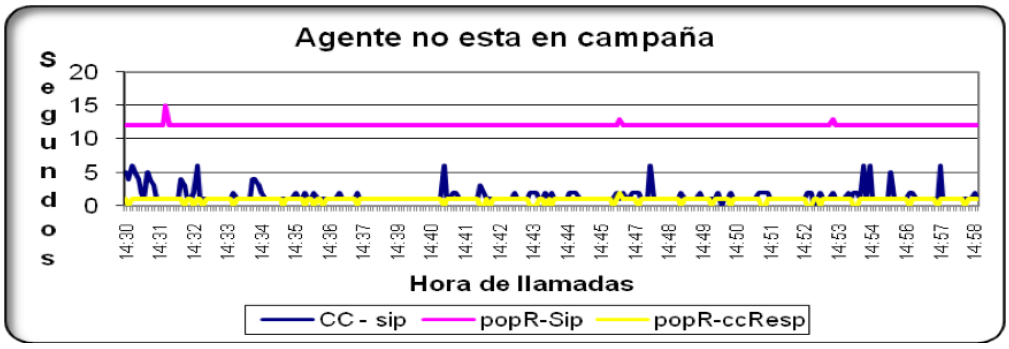
Observando en detalle cada una de las métricas consideradas para esta transacción, se ve que el tiempo entre que la llamada llega al agente y el evento *Call Campaign* está en un entorno de los cuatro segundos. El tiempo entre que la llamada se produce y los datos del cliente llegan al agente aumenta a 14 segundos. El tiempo entre que la respuesta del evento *Call Campaign* es obtenida y la respuesta del *popup* es obtenida, se encuentra entorno a los dos segundos.




**Gráfica 50 - Campaña Automática cinco agentes sin web**

En el caso de la transacción “Agente no está en campaña” los tiempos mostraron un muy pequeño incremento con respecto a su *baseline*. La diferencia de tiempo entre que la llamada llega al agente y se produce el evento *Call Campaign* ahora se encuentra en un entorno de los tres segundos y se obtienen varios picos del entorno de los cinco segundos. El tiempo entre que la llamada llega al agente, y los datos del cliente son mostrados en el monitor de este, sufren un incremento de un segundo con respecto al *baseline*, lo que significa que están en el entorno de los 12 segundos.

La diferencia de tiempo entre las respuestas de *Call Campaign* y *popup* no se ve afectada por esta carga, y registran los mismos tiempos que en el *baseline*. Estos tiempos se muestran en la siguiente gráfica.



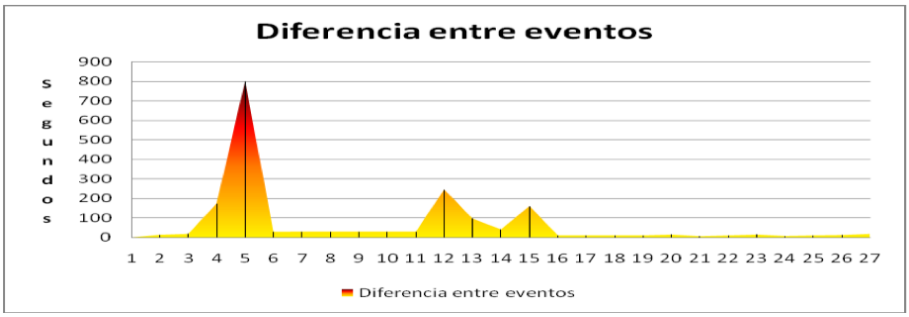

**Gráfica 51 - Agente no está en campaña 5 agentes sin web**

En lo que respecta a la TRN04_IVR se observó que se despacharon sin problemas todas las llamadas realizadas por los clientes.

A continuación, en la Gráfica 52, se presenta el resultado de la ejecución de la consola administrador para esta prueba. En la gráfica se muestra la transferencia de bytes contra el lapso entre la recepción de un evento y otro. Tanto el tiempo como los bytes se dividieron entre 1000 con el fin de poder superponer las dos gráficas y resaltar los resultados.

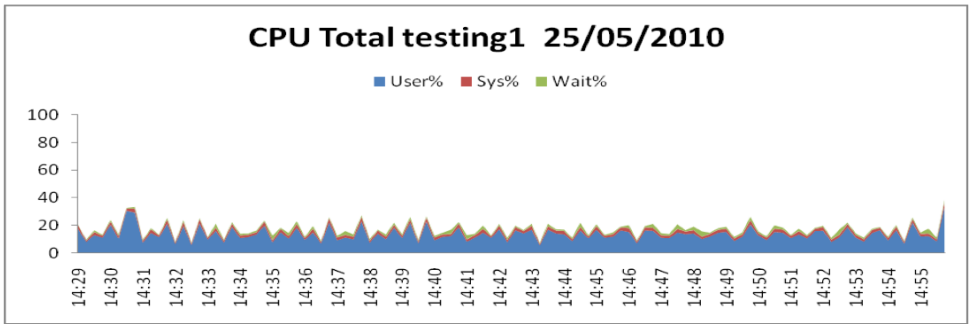

**Gráfica 52 - Administrador cinco Agentes sin web (datos a escala seg/1000, bytes/1000)**

Se observó que la recepción de los paquetes, no es registrada por la herramienta de generación de carga cada un minuto como es el tiempo que esta estipulado demoren en generarse dichos reportes. Al graficar la diferencia de tiempo entre reporte y reporte, siguiente gráfica, queda en evidencia este retraso. Para confirmar que esto se debe a un problema en la generación o envió de reportes fue necesario verificar el comportamiento con un usuario real (testigo) atendiendo una consola de administración, en concurrencia con la ejecución de una prueba.

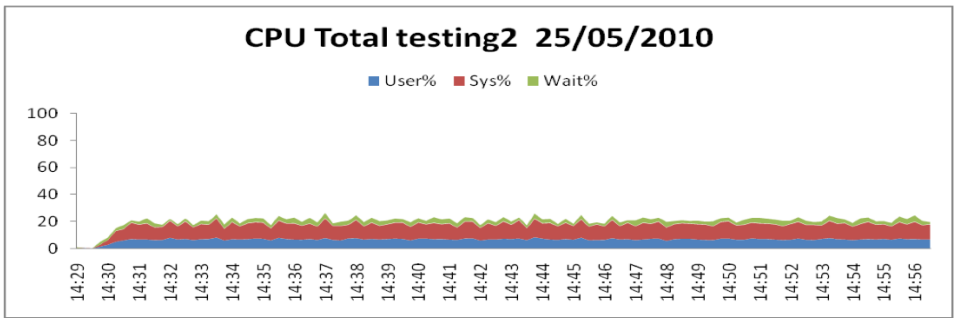



**Gráfica 53 - Diferencia de tiempo de *logs* del Administrador**

Haciendo un análisis del uso del CPU en las máquinas al momento de correr la prueba, se observa que tanto la máquina que corre el IPContact como el servidor Asterisk muestran actividad menor al 20%. Las siguientes gráficas muestran el uso del CPU de cada máquina durante la prueba.




**Gráfica 54 - Uso CPU en la máquina testing1**




**Gráfica 55 - Uso CPU en la máquina testing2**

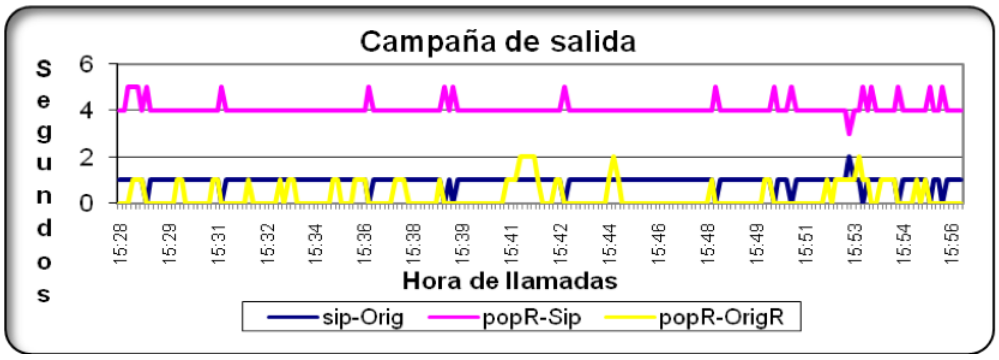
### 4.4.2.3. Cinco agentes con web

Debido a que en la prueba anterior se obtienen respuestas satisfactorias del sistema, y los CPU de las máquinas no se encuentran sobrecargados, se decide incluir las transacciones web en este escenario para observar cómo influyen éstas en la performance total de la prueba. El escenario utilizado en este caso se muestra en la siguiente tabla. El *delay* entre cada ejecución de las transacciones web se configura en tres minutos para permitir que el sistema se recupere entre cada ejecución.

| | 5 Agentes con Web | | | |
|---|---|---|---|---|
| **Transacción** | **Agentes/ usuarios virtuales** | **Duración de la llamada/ transacción (en segundos)** | ***Delay* entre iteración** | **Cantidad de llamadas / iteraciones en una hora** |
| **TRN01_Campaña de Salida** | **5** | **30** | **5** | **514** |
| **TRN02_Campaña Automaticas** | **5** | **30** | **0** | **600** |
| **TRN03_Agente no Esta en Campaña** | **5** | **30** | **0** | **600** |
| **TRN04_IVR** | **5** | **50** | **5** | **327** |
| **TRN05_Administrador** | **5** | **N/A** | **N/A** | **N/A** |
| **TRN06_Datos de Colas** | **1** | **95** | **180** | **13** |
| **TRN07_Actividades de Agentes** | **1** | **21** | **180** | **17** |
| **TRN08_Registro de llamadas** | **1** | **9** | **180** | **19** |

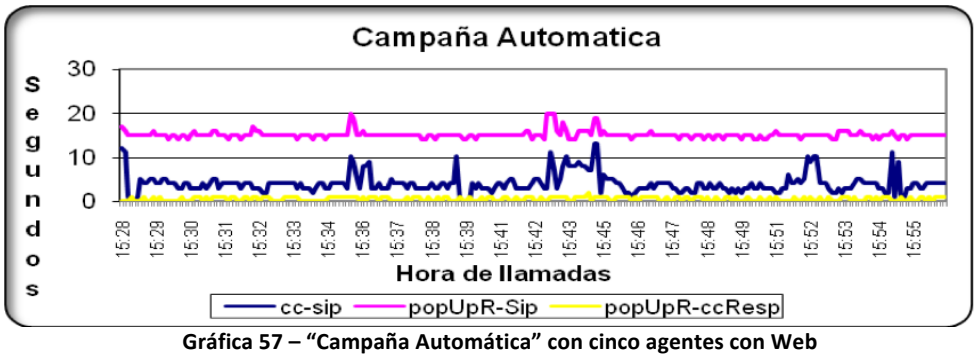
**Tabla 4 - Escenario 5 agentes con web**

Si bien los tiempos de esta transacción se mantienen similares a los tiempos que arrojaron los tiempos base, se puede observar que un conjunto amplio de llamadas tienden a aumentar un segundo las tres métricas que se están considerando. Esto se puede observar en la Gráfica 56.




**Gráfica 56 – "Campaña de salida" con cinco agentes con Web**

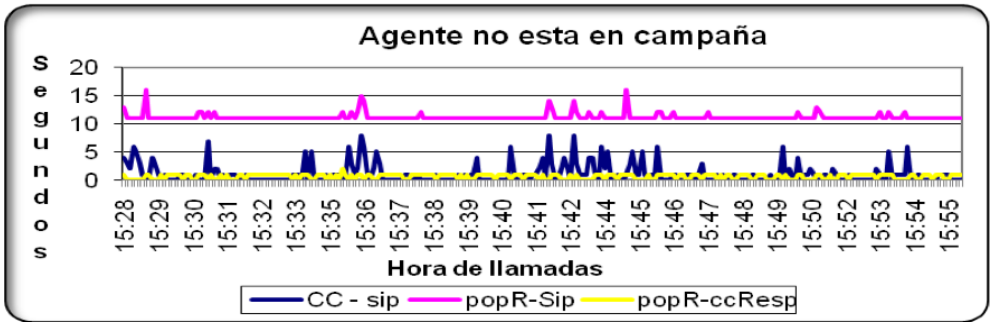
La Gráfica 57 muestra el resultado de la "Campaña Automática".




**Gráfica 57 – "Campaña Automática" con cinco agentes con Web**

La transacción "Agente no está en Campaña" (siguiente gráfica), tiene un comportamiento similar que la transacción anterior. Esto es, se mantienen los tiempos de la última prueba (5 agentes sin web), salvo en ciertos períodos en los cuales se observan una secuencia de picos en los tiempos

de respuesta. Se puede ver que los períodos que se mencionan, se corresponden a los picos de CPU donde los procesos de usuario (color azul) superan el 70% de uso.

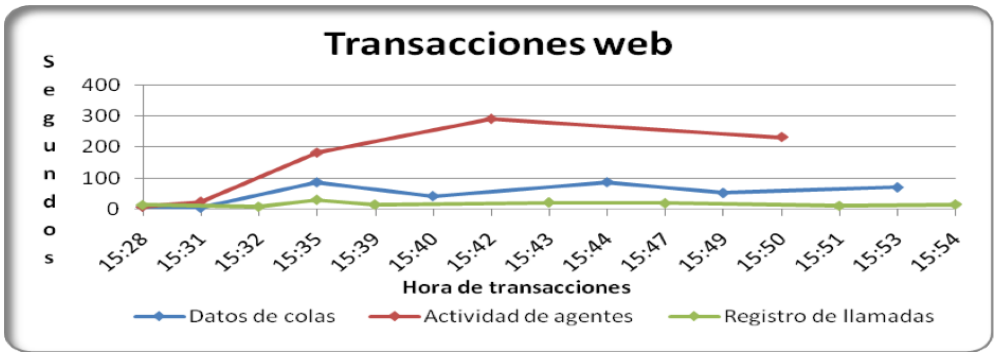



**Gráfica 58 – "Agente no está en campaña" con cinco agentes con web**

Se pueden identificar tres lapsos donde se degradan los tiempos de respuesta. Dichos períodos son los siguientes: [15:35,15:37], [15:42,15:46], [15:51,15:55]. Al observar la Gráfica 60, que muestra el uso del CPU en la máquina que ejecuta el IPContact, se ve que en estos mismos períodos el procesador se encuentra saturado con un 100% de su uso, lo que explica la degradación en los tiempos de respuesta obtenidos. Sin considerar estos picos de tiempo, el resto de la prueba se mantiene con tiempos muy similares a la prueba anterior donde no se incluyen las consultas web.

Para la TRN04_IVR se despachan todas las llamadas generadas por los clientes sin inconveniente.

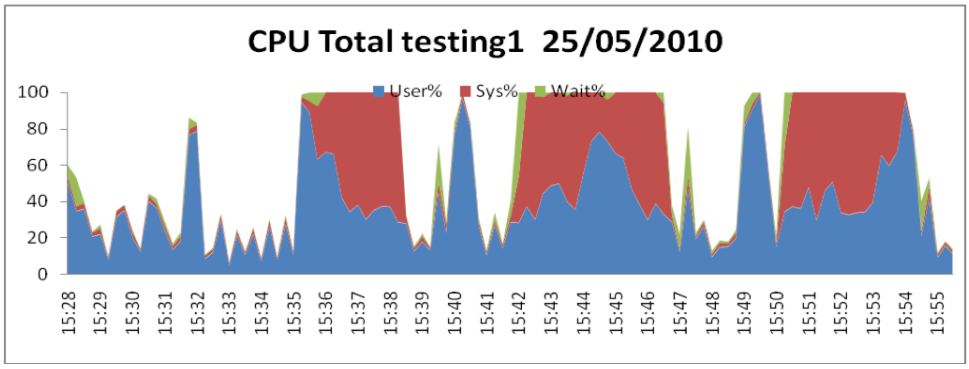
A continuación se hace un análisis de las transacciones web para este escenario. La Gráfica 59 muestra los resultados, se marcan con puntos el momento en que se ejecutan las consultas de modo facilitar esta observación.




**Gráfica 59 - Tiempo de transacciones web**

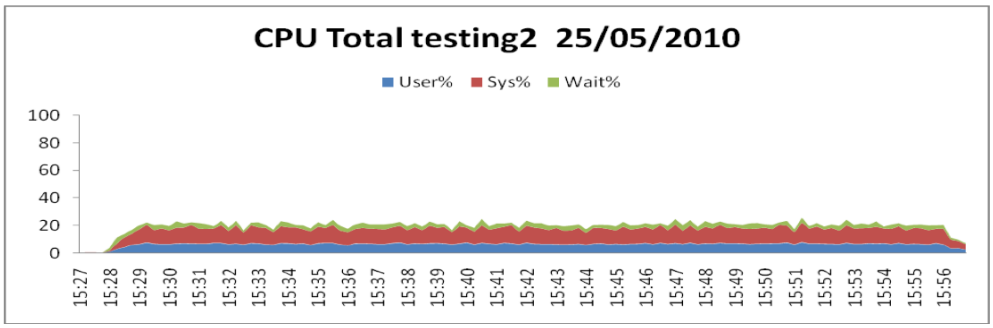
Se observa que tanto la transacción "Datos de Colas" como "Registro de Llamadas", mantienen tiempos del mismo orden que en los tiempos base, mientras que la transacción "Actividad de Agentes" aumenta considerablemente con respecto al tiempos base.

Observando el estado del procesador de la máquina en la cual corre IPContact, se puede verificar la correspondencia entre los picos de uso del CPU (procesos de usuario), que se muestran en la Gráfica 60, y el momento en que se ejecutan las transacciones "Datos de Colas" y "Actividad de Agentes".




**Gráfica 60 - Uso CPU en la máquina testing1**

La siguiente gráfica, muestra el uso de CPU de la máquina donde ejecuta el Asterisk, no presentó un incremento en el uso del procesador con respecto a la ejecución del escenario anterior, y se mantiene estable sin superar un 20% durante todo el transcurso de la prueba.




**Gráfica 61 - Uso de CPU en la máquina testing2**

Se concluye que las transacciones web "Datos de Colas" y "Actividad de Agentes" saturan el procesador produciendo un retraso en los tiempos de respuesta de las transacciones (sobre Asterisk) "Campaña Automática" y "Agente no está en campaña".

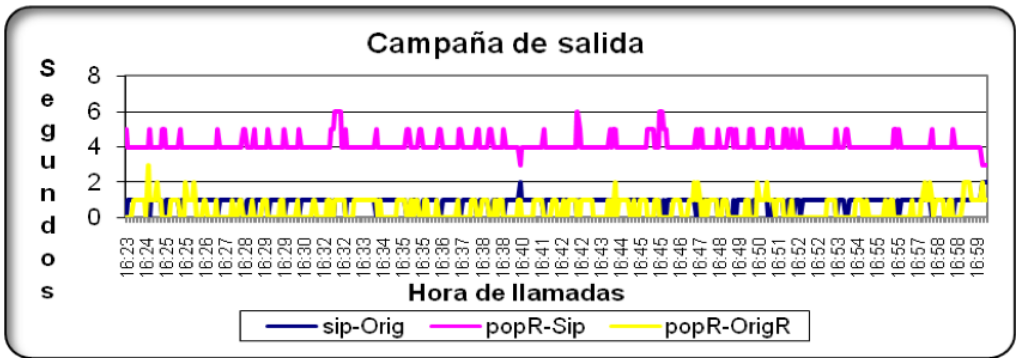
#### 4.4.2.4. 10 Agentes sin Web

Esta prueba surge con el objetivo de obtener más datos sobre el rendimiento del Asterisk. Siguiendo esto, se crea el escenario de la siguiente tabla, en el cual se quitan las consultas web y se escala en cuanto a la cantidad de agentes involucrados en las transacciones que utilizan Asterisk.

| | 10 Agentes Sin Web | | | |
|---|---|---|---|---|
| **Transacción** | **Agentes/ usuarios virtuales** | **Duración de la llamada / transacción (en segundos)** | ***Delay*** **entre iteración** | **Cantidad de llamadas / iteraciones en una hora** |
| **TRN01_Campaña de Salida** | **10** | **30** | **5** | **1028** |
| **TRN02_Campaña Automaticas** | **10** | **30** | **0** | **1200** |
| **TRN03_Agente no Esta en Campaña** | **10** | **30** | **0** | **1200** |
| **TRN04_IVR** | **10** | **50** | **5** | **654** |
| **TRN05_Administrador** | **10** | **N/A** | **N/A** | **N/A** |
| **TRN06_Datos de Colas** | **0** | **N/A** | **N/A** | **N/A** |
| **TRN07_Actividades de Agentes** | **0** | **N/A** | **N/A** | **N/A** |
| **TRN08_Registro de llamadas** | **0** | **N/A** | **N/A** | **N/A** |

A continuación en la Gráfica 62, se muestran los tiempos obtenidos para la ejecución de la "Campaña de Salida". Al compararlo con el escenario de "5 Agentes sin web" (Gráfica 49), se ve que el único tiempo que se incrementa es la diferencia entre que se inicia la llamada y los datos del cliente se muestran en la pantalla del Agente. Este tiempo en el escenario mencionado se

encuentra en un entorno de los cuatro segundos, mientras que en este nuevo escenario la mayoría de las llamadas presentan un tiempo de cinco segundos y en algunos casos seis segundos.



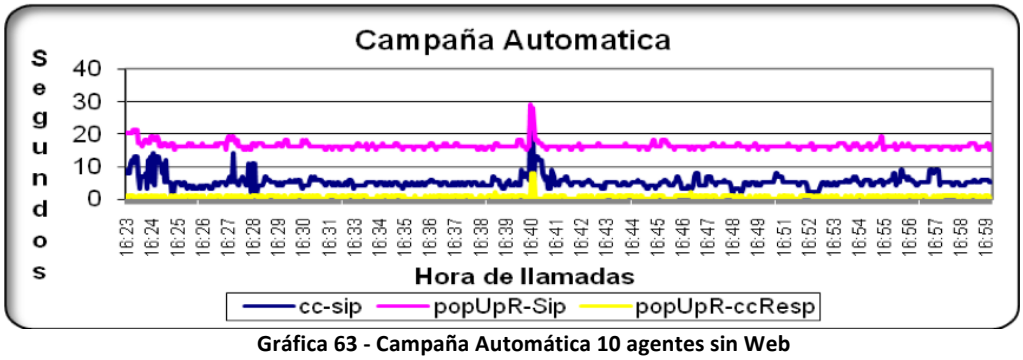

**Gráfica 62 - Campaña de Salida 10 agentes sin Web**

En la siguiente gráfica se muestra el resultado de la "Campaña Automática" para este escenario. Se observa un incremento de los tiempos de respuestas con respecto al escenario "5 Agentes sin Web". Si se considera la diferencia de tiempo entre que la llamada llega al agente y el evento *Call Campaign*, se ve que los tiempos de respuesta están en un entorno de los seis segundos, con varios picos de 10 y 12 segundos, a diferencia el escenario "Cinco Agentes sin Web" que este tiempo se mantiene estable en cuatro segundos.

Lo mismo sucede con la diferencia de tiempo entre que la llamada se produce y los datos del cliente se muestran en el monitor del Agente. Esta medida alcanza un entorno de los 18 segundos y en algunos casos 20 segundos.

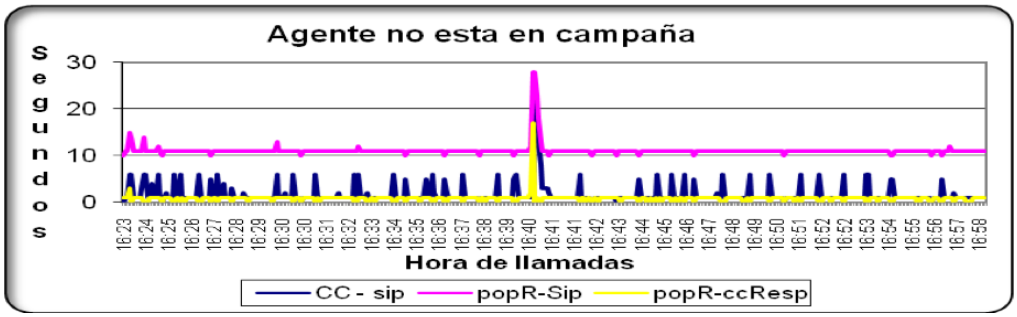
El tiempo entre que la respuesta del evento *Call Campaign* es obtenida y la respuesta del *popup* es obtenida se mantiene estable con respecto al escenario antes mencionado.

A las 16:40 de esta prueba se observa un pico en los tres tiempos. Este evento lo produjo la generadora de carga, ya que en ese momento alcanza un 100% del uso de su CPU. Por lo tanto este pico de tiempo no es considerado como parte del problema. La Gráfica 65 muestra el uso del CPU en la generadora de carga.




**Gráfica 63 - Campaña Automática 10 agentes sin Web**

A continuación se muestra la transaccion "Agente no está en campaña". Aquí se observa un incremento de unos dos segundos en la diferncia de tiempo entre que se genera la llamada y el evento *Call Campaign* es disparado. El resto de los tiempos se mantienen estables, si lo comparamos con el escenario "5 Agentes sin web".

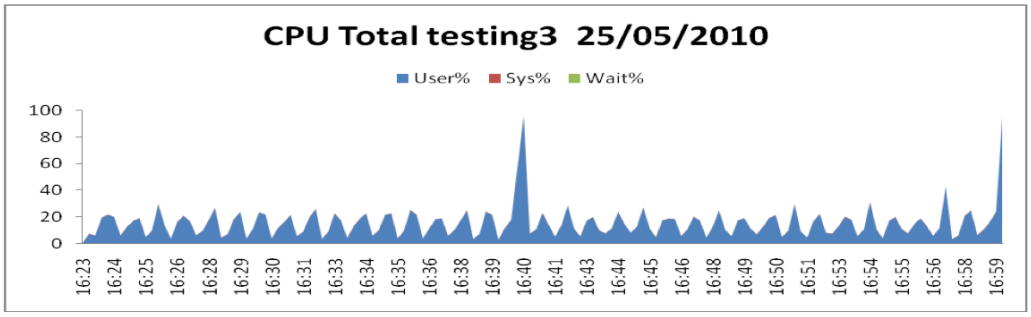



**Gráfica 64 - Agente no está en Campaña 10 agentes sin Web**

No se detectaron inconvenientes en las llamadas generadas en la TRN04_IVR.

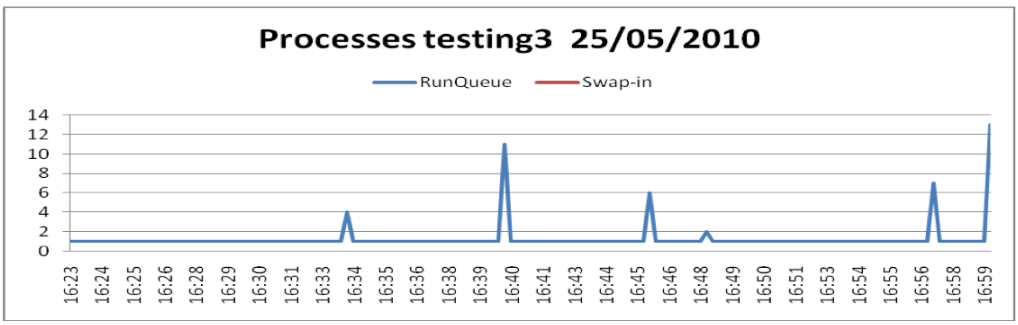
El uso del CPU durante la prueba, de las máquinas que alojan el sistema (testing1 y testing2), se incremento en un 10% respecto al escenario "5 Agentes sin web".

La siguiente gráfica muestra el uso del CPU en la generadora de carga para esta prueba. Esta máquina se mostró holagada durante toda la prueba, salvo en un entorno a las 16:40, que presenta un pico de uso del CPU.




**Gráfica 65 - Uso de CPU en la máquina testing3**

Observando la siguiente gráfica, donde se muestra el estado de la cola de procesos del sistema, se puede ver la correspondencia con el pico.




**Gráfica 66 - Cola de procesos en la máquina testing3**

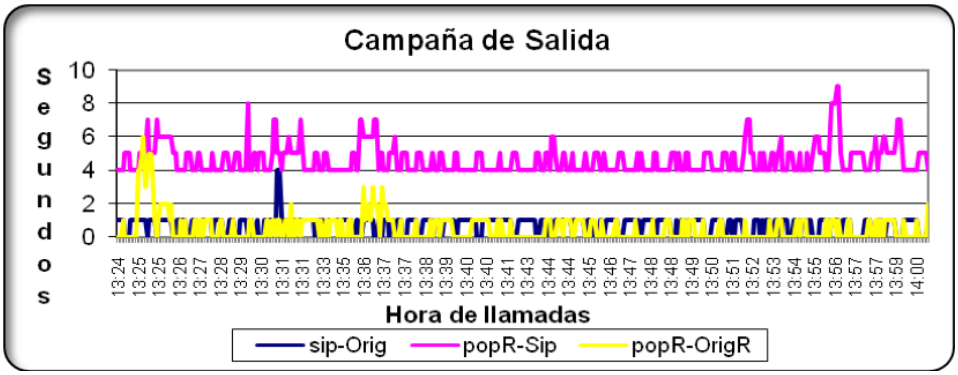
#### 4.4.2.5. 10 Agentes con web

El siguiente escenario consiste en agregar las consultas web al escenario anterior.

| Transacción | 10 Agentes con web | | | |
|---|---|---|---|---|
| | Agentes/ usuarios virtuales | Duración de la llamada/ transacción (en segundos) | *Delay* entre iteración | Cantidad de llamadas / iteraciones en una hora |
| **TRN01_Campaña de Salida** | **10** | **30** | **5** | **1028** |
| **TRN02_Campaña Automaticas** | **10** | **30** | **0** | **1200** |
| **TRN03_Agente no Esta en Campaña** | **10** | **30** | **0** | **1200** |
| **TRN04_IVR** | **10** | **50** | **5** | **654** |
| **TRN05_Administrador** | **10** | **N/A** | **N/A** | **N/A** |
| **TRN06_Datos de Colas** | **1** | **95** | **180** | **13** |
| **TRN07_Actividades de Agentes** | **1** | **21** | **180** | **17** |
| **TRN08_Registro de llamadas** | **1** | **9** | **180** | **19** |

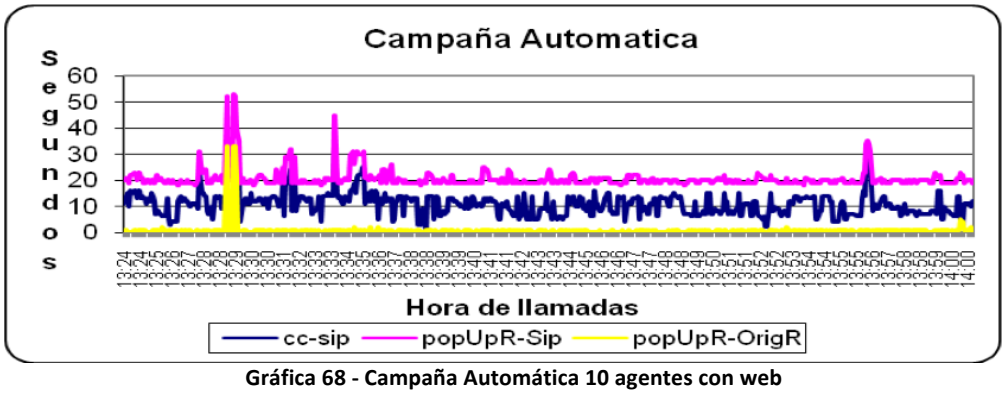
**Tabla 5 - Escenario 10 Agentes con web**

Observando la Gráfica 67 que muestra el resultado de la "Campaña de Salida", se puede ver que los tiempos no varían considerablemente con respecto a la prueba anterior, que consistió en la misma cantidad de Agentes, pero que no incluía las consultas web. La diferencia de tiempo entre que se genera la llamada y la llamada llega al agente, se mantiene en el orden del segundo. La diferencia entre que se inicia la llamada y los datos del cliente se muestran en la pantalla del agente es de cuatro a cinco segundos, mientras que el tiempo transcurrido entre que se obtiene la respuesta del *originate* y se obtiene la respuesta del *popup*, se mantiene también en el entorno de un segundo.




**Gráfica 67 - Campaña de salida 10 agentes con web**

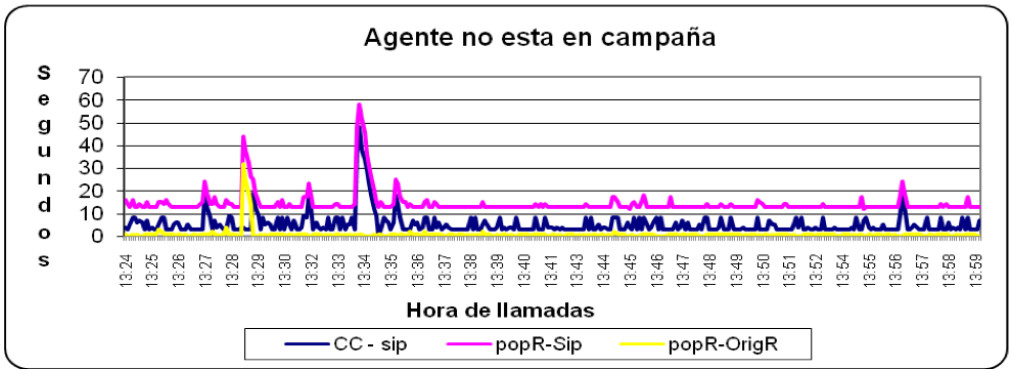
En el caso de la "Campaña Automática", que se muestra en la siguiente gráfica, los tiempos se incrementan con respecto a la prueba anterior. Se observó que la mayor repercusión se produce en el tiempo entre que se establece la llamada y se produce el evento *Call Campaign*. Este tiempo se incrementa en el orden de los cinco segundos en este caso. El resto de los tiempos se incrementan en menor orden.




**Gráfica 68 - Campaña Automática 10 agentes con web**

La siguiente gráfica muestra el resultado de la transacción “Agente no está en campaña”. Se observa un incremento en los tiempos con respecto al escenario anterior.

La diferencia de tiempo entre que se genera la llamada y se produce el *Call Campaign* registra un promedio de cinco segundos. La diferencia de tiempo entre que la llamada se produce y los datos del cliente se muestran en el monitor del Agente, también aumenta, registrando en este caso tiempos del entorno de los 15 segundos, mientras que la diferencia entre las respuestas del evento *Call Campaign* y la respuesta del *popup* se mantienen en el orden de los dos segundos como en el escenario anterior.




**Gráfica 69 - Agente no está en campaña 10 agentes con web**

El resultado de la TRN04_IVR fue similar a los casos anteriores, todas las llamadas generadas por clientes fueron despachadas sin problemas.

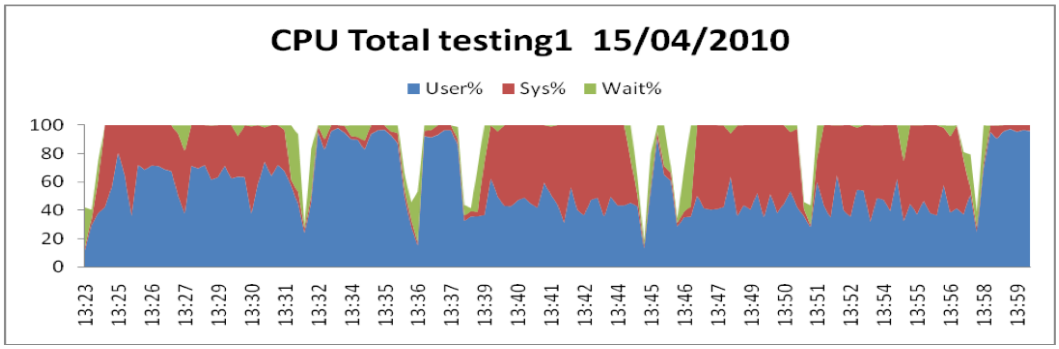
En la Gráfica 70, se muestran los tiempos de respuestas de las transacciones web para este escenario. A partir de 13:32 “Datos de colas” obtiene solamente respuestas de error, por lo tanto no registra tiempos. Se observó que los tiempos sufren un incremento:

- “Datos de colas” registra tiempos de 50 a 150 segundos.
- “Actividad de agentes” de 50 a 450 segundos.
- “Registro de llamadas” entre 5 y 50 segundos.

Las variaciones en los tiempos están ligados a si el conjunto de datos que se utiliza en cada ejecución de la consulta se encuentra o no en memoria cache.

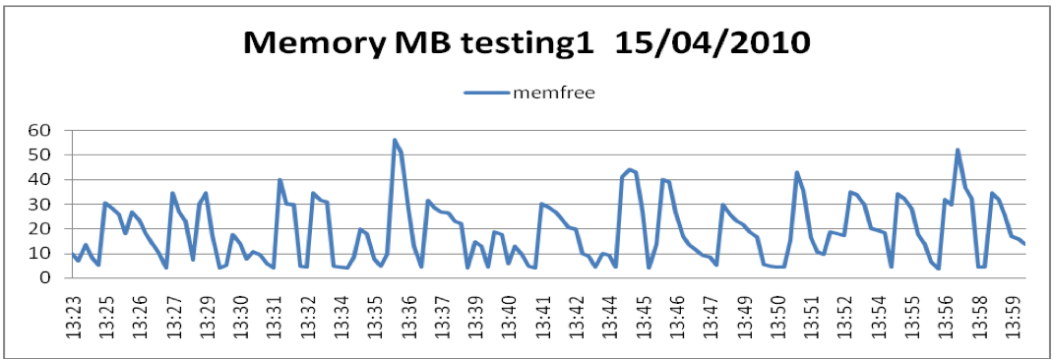

**Gráfica 70 - Tiempo de transacciones web**

Observando el uso del CPU de la máquina testing1 durante esta prueba, siguiente gráfica, se observa que el mismo se encuentra en un orden del 100% la mayor parte del tiempo. Hay una relación entre algunos picos de 100% de uso del procesador con procesos de usuario y el momento que se ejecuta la transacción “Actividad de Agentes”.




**Gráfica 71 - Uso del CPU en la máquina testing1**

La siguiente gráfica muestra que la máquina prácticamente no tiene memoria.




**Gráfica 72 - Memoria libre en la máquina testing1**

En el caso de la máquina testing2 se observó al momento de la ejecución, que el CPU se mantuvo estable, en un entorno del 30% de su uso durante toda la prueba.

### 4.4.3. Conclusiones

La primera conclusión extraída de las pruebas realizadas es que las transacciones web consumen excesivos recursos de CPU. Esto se detecta desde el inicio al ejecutar los tiempos base. Estas transacciones consumen la memoria del servidor IPContact, lo que implica que el procesador dedique muchos ciclos de CPU al proceso de paginación, y agote este recurso. Como consecuencia de este excesivo consumo, se decidió no escalar la carga en este aspecto, ya que las repercusiones de estas transacciones sobre el resto del sistema impedían analizar con propiedad el desempeño de los restantes componentes de la mezcla.

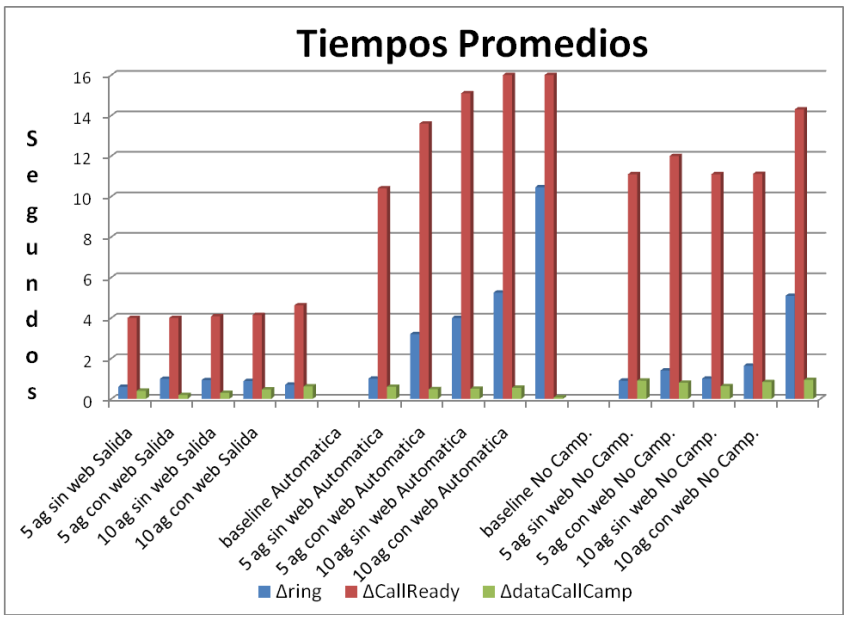
Para volver a ejecutar escalando la carga para las transacciones web, es necesario aumentar los recursos de hardware o reconfigurar la plataforma, hasta que despache los *baselines* sin saturar.

Las consultas web que se ejecuten con datos que se encuentran en memoria cache del DBMS tienen tiempos de respuesta hasta tres veces más bajos.

No se observó saturación en el sistema ejecutando únicamente transacciones SIP. En caso de desear encontrar el punto de quiebre del Asterisk, si bien no era el objetivo primario de esta prueba, se deberán crear una cantidad mayor de agentes.

Se observó que al encontrarse saturado el CPU por las transacciones web, se degradan los tiempos de respuesta del resto de las transacciones salvo en la "Campaña de Salida", la cual no se vio afectada con el consumo de CPU mencionado.

La siguiente gráfica se muestra la evolución de los tiempos de respuesta de las transacciones Asterisk en las distintas pruebas ejecutadas a lo largo del proyecto.




**Gráfica 73 - Evolución de los tiempos en las distintas pruebas**

La gráfica anterior muestra que en cada iteración (a medida que se aumenta la carga generada), se incrementa la diferencia de tiempo entre que una llamada es generada y se presentan los datos del cliente en la pantalla del agente (*popup*). Este lapso es importante ya que implica que un agente interactúe (durante este tiempo) con un cliente del cual no conoce sus datos.

Es de destacar que, más allá de las pruebas realizadas, se dejó configurada en IPContact una plataforma para continuar ejecutando pruebas. La cual permite modelar usos esperados en un cliente dado, simular su operativa y verificar la performance en la infraestructura a instalarse y bajo una carga determinada previo a poner el sistema en producción.

# 5. Aplicación en la salud

El tercer caso es sobre el sistema Geosalud, desarrollada con Genexus 9 generando en Java 3 capas. Dicho proyecto se realizó entre los meses de Noviembre de 2012 a Marzo de 2013 en instalaciones de Geocom (fase de preparación y ejecución) y ASSE (fase inicial de ejecución).

La prueba de performance se basó en un estudio del escenario estimado de uso de la aplicación, por lo tanto para la carga esperada en el sistema se utilizó como referencia la operativa que se llevaba adelante en ASSE. Como consecuencia de esto, se analizaron más de seis transacciones o ciclos funcionales. Algunas de las cuales incluían varios ciclos funcionales, motivo por el cual se separaron.

Al igual que los casos anteriores, este capitulo consta de una descripción detallada de las pruebas realizadas, incluyendo los escenarios definidos, el ambiente de prueba y las métricas a recopiladas. La primera sección, describe las pruebas desde el punto de vista de sus objetivos, tanto en cuanto al software que se probó como al objetivo específico de la prueba (ver "Objetivo"). Se repasa y verifica, a grandes rasgos, que la metodología utilizada para llevar a cabo el proyecto es la presentada anteriormente. La sección "Escenario" describe todas las características de las pruebas desde el punto de vista del entorno de las mismas. Ítems como infraestructura utilizada y escenarios de carga son vistos en dicho apartado. La sección "Automatización y armado del ambiente de pruebas" describe aquellos temas relativos a la robotización de la prueba, punto neurálgico en la realización de una prueba de performance. En "Monitorización" se describen las herramientas utilizadas para realizar dicha actividad. "Ejecución de pruebas" describe las actividades en donde se ejecutaron los scripts antes definidos y se monitoreo la infraestructura. Finalmente, en "Conclusiones" se realizará un recuento de los resultados obtenidos y los pasos a seguir en el futuro para lograr mejoras en la performance del sistema Geosalud.

## 5.1. Descripción de las pruebas

En esta sección se describirá el objetivo del ensayo realizado y el software sobre el que se realizó el mismo. Estos puntos pondrán en contexto a estas pruebas y explicarán muchas de las decisiones tomadas durante el proyecto.

### *5.1.1. Software Probado*

El software que se sometió carga fue Geosalud, en versiones desde 1.7.10b correspondiente a la fecha 10 de diciembre del 2012 hasta la versión 1.8 correspondiente a la fecha de marzo del 2013. Los cambios en versiones obedecieron a la imposibilidad de las primeras versiones de soportar la carga planificada y a las evoluciones que se realizaron sobre el mismo.

Se trata de una aplicación en tres capas con datos (DBMS), lógica (servidor de aplicaciones) y presentación (cliente fino, navegador Firefox de los usuarios), desarrollado en Genexus 9_0_6-019, generando código Java contra bases de datos MariaDB 10.0.

El sistema trabaja con datos que el sistema de la admisión de pacientes genera. Esto implica que no se crean órdenes de trabajo en el sistema si no pasan por el sistema de admisión; si bien esto da un nivel de integración al momento de la definición de la prueba, no era el foco probar este sistema, con lo cual se trabajará con datos generados de manera previa al inicio de las pruebas.

| | |
|---|---|
| **Software** | Geosalud |
| **Plataforma** | 3 Capas |
| **DBMS** | MariaDB – 10.0 |
| **Lenguaje Desarrollo** | Genexus 9_0_6-019 generando Java. |

### 5.1.2. Objetivos

Esta prueba se desarrolla con el objetivo de verificar la performance del sistema y estudiar el comportamiento del mismo ante cargas de uso para las cuales el sistema fue diseñado. En pocas palabras, verificar que los requisitos no funcionales de performance del sistema puedan cumplirse y que el software es apropiado para las necesidades de ASSE.

Este estudio implica una serie de actividades y de verificaciones a realizarse, entre las cuales se pueden enumerar:

- Tiempos de respuesta de los programas del sistema. Estos tiempos pueden ser agrupados en transacciones o sectores del ciclo funcional.
- Desempeño de los servidores de aplicación.
- Desempeño del servidor de bases de datos.

La verificación de estos parámetros cumple la finalidad de determinar el uso de la infraestructura, de manera que dichos datos puedan ser utilizados en el *sizing* de futuras implantaciones del sistema y acompasar los recursos asignados a la implantación al plan de puesta en producción existente. A su vez, esto permitirá detectar, identificar y solucionar problemas de performance (sin necesidad de ejecutar sobre la plataforma definitiva de producción).

Lo antes descripto permitirá predecir comportamiento bajo carga en el ambiente que se monte para las pruebas, detectar y solucionar problemas de performance y tener un primer acercamiento al comportamiento que tendrá finalmente en producción (ayudar en el *sizing* de plataformas e identificar patrones de comportamiento que pueden ser útiles a la hora de administrar la aplicación). Sobre esta infraestructura se intentará definir, también, el punto de "quiebre" del sistema.

Los requisitos no funcionales del sistema fueron establecidos en función de la cantidad de operaciones y usuarios nominales del sistema en determinado lapso, establecido como la ventana de tiempo de las pruebas "una hora".

## 5.2. Escenario

Se describe a continuación los escenarios bajo los cuales se desarrollaron las pruebas. Estos escenarios se componen tanto del entorno de infraestructura que se utilizó como de los distintos escenarios de carga que se reprodujeron con la herramienta de generación de carga.

Estas combinaciones marcan la validez de las pruebas realizadas y de los resultados obtenidos en las mismas.

### 5.2.1. Escenario de Infraestructura

Para la realización de las pruebas se comenzó utilizando una infraestructura parecida a producción, Figura 14; la infraestructura de producción se encuentra esquematizada en la Figura 15. Luego de las pruebas iniciales, se decidió realizar el resto de las pruebas en la infraestructura virtualizada disponible en Geocom. Esta infraestructura demostró ser suficiente y permitió avanzar con las pruebas con una velocidad que no era posible realizarla en el entorno de ASSE. Debido a que el foco de la prueba devino en la mejora de la aplicación previo al análisis de la infraestructura, se definió este nuevo enfoque como el aceptable.

En las diferentes descripciones de la infraestructura se puede observar también el esquema de simulación. Se puede notar la existencia de diferentes roles en los equipos utilizados:

- **Generadores de carga:** Son los responsables de generar la carga (pedidos HTTP) de cada uno de los usuarios virtuales que participan en la prueba.

- **Balanceadora de carga:** Es el responsable de balancear la carga el procesamiento de los pedidos HTTP realizados por los usuarios virtuales que se incluyen en la prueba. La política de balanceo es *round robin* (un pedido a cada servidor).
- **Servidores de aplicación:** Son los responsables de realizar el procesamiento de los pedidos HTTP realizados por los usuarios virtuales que se incluyen en la prueba.
- **Servidor de Base de Datos:** Servidor oficiando de host para el DBMS que contiene los datos del sistema. En producción la conexión con el servidor de aplicación es a 1Gbps. En la plataforma de test están conectados con un *switch* de 100Mbps.

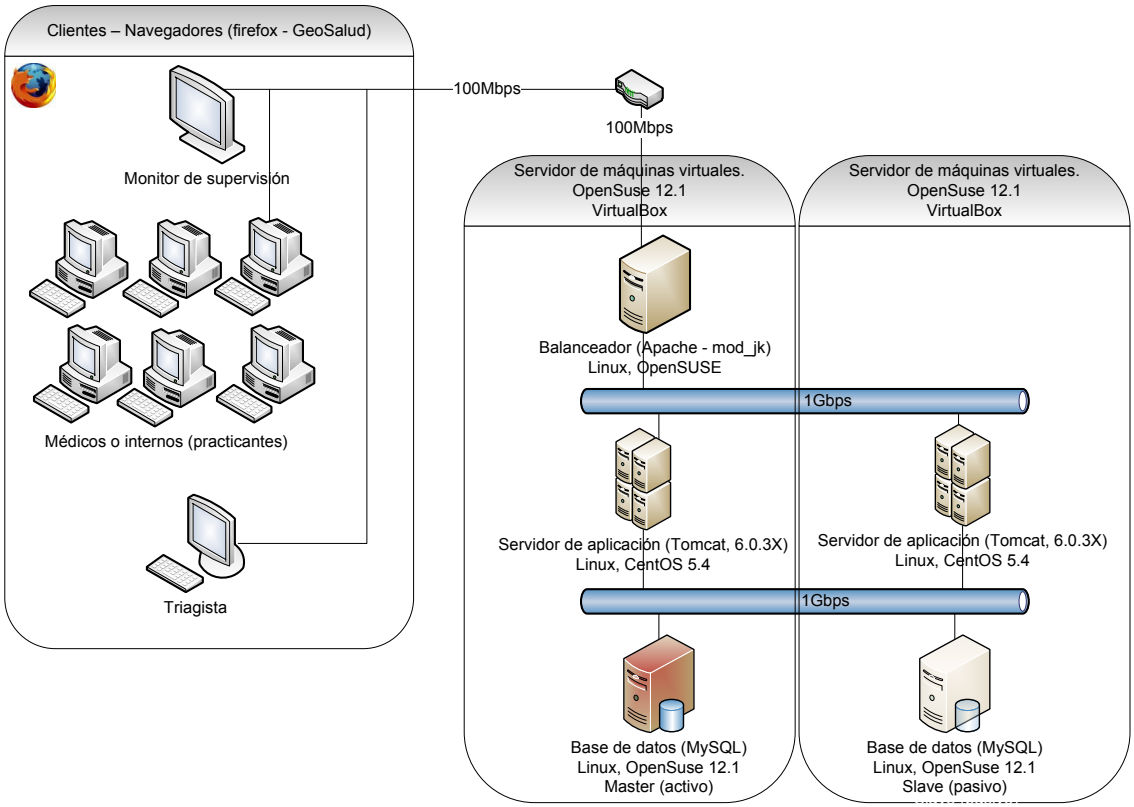



**Figura 14: Infraestructura de producción**

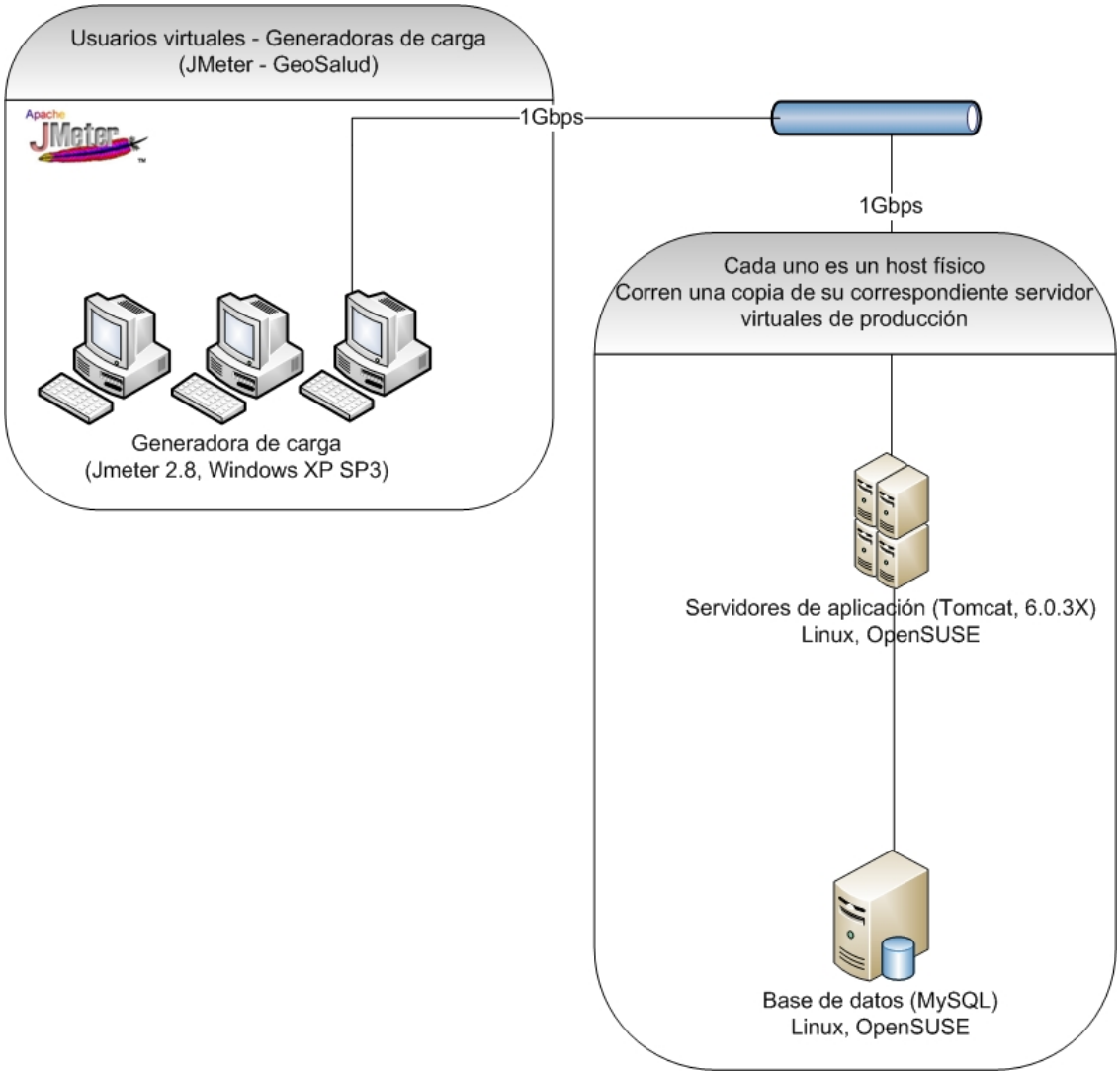



**Figura 15: Infraestructura de pruebas - ASSE**

Bajo esta infraestructura se realizaron distintas pruebas ejercitando distintos escenarios, principalmente las primeras pruebas de tiempos base. Luego, por problemas de desempeño de ciertas tareas necesarias para la ejecución de las pruebas (por ejemplo, realización de recuperación de las bases de datos) y la posibilidad de trabajar directamente acompañados de los desarrolladores, se decidió continuar con las pruebas en la infraestructura disponible por Geocom.

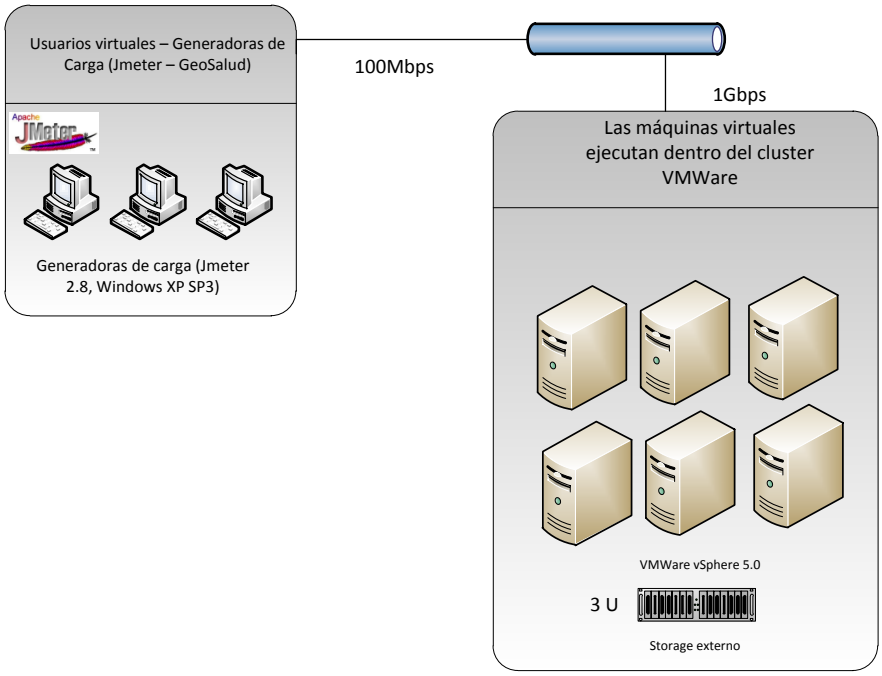



**Ilustración 1: Infraestructura de pruebas - Geocom**

En la siguiente sección se describirán características de hardware y software de los equipos antes presentados.

#### 5.2.1.1. Hardware y Software de Base

Las características de los equipos utilizados en los ambientes de Testing (Geocom y ASSE) son las siguientes:

**Generadora de Carga**

| Hardware |
|---|
| Modelo Dell Inspiron 6400 |
| CPU Core 2 Duo 1.7Hz |
| Memoria 2GB |
| Disco SATA 80GB 5400 RPM |

| Software |
|---|
| Windows XP Professional SP3 |
| Java 1.7.0_09 |
| JMeter 2.9 (con extensiones implementadas por CES) |

La generadora de carga se conecta en redes LAN "switcheadas" de 1Gbps en el ambiente de ASSE y 100Mbps de Geocom.

**Equipos servidores para pruebas**

En el ambiente de ASSE, los cuatro hosts físicos que pertenecen al SUT corren una copia exacta de cada una de las máquinas virtuales de producción: un balanceador, dos contenedores, una base de datos. Estos host cuentan con características similares y son las siguientes:

| Hardware |
|---|
| Marca / Modelo: Equipos clon. |
| CPU: AMD Phenom II X6 1055T |
| Memoria: Memoria 8GB |
| Discos: 500 GB HDD (Raid1) |
| Red: 2xGbE |

En el ambiente de Geocom, las máquinas virtuales se encuentran basadas en VMWare vSphere 5.0 enterprise, dentro de un *cluster* de producción que Geocom tiene disponible para ejecutar su infraestructura. Dentro de dicho *cluster* (que cuenta con un total de 28 cores Intel Xeon de 2.4 GHz y 323GB de RAM totales), de definieron dos máquinas virtuales con las mismas características de las que se encuentran en ASSE. Estas máquinas virtuales se encuentran en un *storage* IBM Dichas máquinas fueron ejecutadas siempre en hosts diferentes, de manera que no se eliminara en ningún momento la influencia de los tiempos de red entre los dos componentes del SUT.

**Servidor de aplicación**

| Software |
|---|
| Sistema Operativo: Linux – OpenSuse 12.1 |
| Contenedor: Apache Tomcat 6.0.3X |

**Servidor de base de datos** (1)

| Software |
|---|
| Sistema Operativo: Linux – OpenSuse 12.1 |
| DBMS: MySQL |

En producción se cuenta con dos equipos físicos para la virtualización con las siguientes características:

**Servidores de virtualización** [***vb01 (192.168.1.120); vb02 (192.168.1.121)***]

| Hardware |
|---|
| Marca / Modelo: HP DL380G6 |
| CPU: 24 cores Intel(R) Xeon(R) 3.07GHz |
| Memoria: 64GB de RAM |
| Discos: 3 discos SATA de 60GB c/u |
| Red: 2 placas Intel 82576 Gigabit |

| Software |
|---|
| Sistema Operativo: OpenSuse 12.1 |
| Plataforma de Virtualización: VirtualBox |

**Balanceador (1)**

| Software |
|---|
| Sistema Operativo: Linux – CentOS 5.4 |
| Software de Balanceo: Apache mod_jk |

**Servidor de aplicación**

[uno en cada Servidor Físico: ***web07.produccion (10.200.31.127), web08.produccion (10.200.31.128)***]

| Hardware |
|---|
| VirtualBox |
| CPU: 4 cores @ 3.07GHz |
| Memoria: 5GB de RAM |
| Disco: 15GB |
| Red: 1 interfaz |

| Software |
|---|
| Sistema Operativo: CentOS 5.4 |
| DBMS: MySQL |

| Configuración |
| --- |
| /usr/java/jdk1.6.0_20//bin/java<br>-Djava.util.logging.config.file=/var/tomcat/Instancia-GS/conf/logging.properties<br>-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Xms2048m<br>-Xmx4096m -Xss4M -XX:MaxPermSize=4096m -XX:PermSize=2048m -XX:+UseParallelOldGC<br>-Djava.endorsed.dirs=/var/tomcat/Instancia-GS/endorsed<br>-classpath /var/tomcat/Instancia-GS/bin/bootstrap.jar<br>-Dcatalina.base=/var/tomcat/Instancia-GS -Dcatalina.home=/var/tomcat/Instancia-GS<br>-Djava.io.tmpdir=/var/tomcat/Instancia-GS/temp org.apache.catalina.startup.Bootstrap<br>start |

**Servidor de base de datos**

[2 activo / pasivo, se detalla solo el activo – el pasivo tiene 40Gb menos de disco]

| Hardware |
| --- |
| VirtualBox |
| CPU: 2 cores @ 2.66GHz |
| Memoria: 2GB de RAM |
| Disco: 60GB |
| Red: 1 interfaz |

| Software |
| --- |
| Sistema Operativo: OpenSuse 12.1 |
| DBMS: MySQL |

### *5.2.2.Escenario de Carga*

Sobre el escenario de infraestructura establecido, se definió el escenario de carga que se detalla en esta sección.

Este escenario está compuesto por:

- Transacciones que participan en la prueba.
- Usuarios del sistema.
- Cantidad de usuarios que ejecutan cada una de las transacciones.
- Cantidad de transacciones que se ejecutan en una hora (*Throughput*) para cada transacción.

El escenario fue definido en conjunto con Geocom y ASSE, basado en estadísticas del uso del sistema actual y del uso previsto del sistema a implantar. Para su definición, se tuvieron en cuenta:

- Cantidad de ejecuciones de las transacciones.
- Cantidad de usuarios concurrentes.
- Importancia de cada transacción en la operativa.
- Consumo de recursos estimado de la transacción.
- Realidad actual de uso de los centros del piloto de ASSE.

El escenario corresponde al uso del sistema por parte de los 46 hospitales del ASSE. De dicho escenario (conocido de ahora en adelante como escenario del 100% de carga) se definieron escenarios intermedios con porcentajes de cargas menores. Este punto es fundamental para poder desprender los requerimientos de hardware en los diferentes escenarios y el punto de quiebre del sistema.

#### 5.2.2.1. Transacciones seleccionadas

Se seleccionaron seis transacciones interactivas de la operativa diaria de ASSE para el escenario definido. Sin embargo se hizo un estudio de todas las transacciones que se ejecutan sobre el sistema a fin de generar un nivel de carga similar al de producción. Producto de este análisis se llegó a que la carga generada por las transacciones seleccionadas corresponde al 49% de la carga total, con lo cual el restante 51% en la simulación será la extrapolación proporcional de las transacciones seleccionadas.

A continuación se enumeran las transacciones interactivas del escenario diurno:

| Transacción | Descripción |
|---|---|
| **TRN01_VISOR [CONSULTA_ORDENES_SERVICIO]** | Esta transacción se ejecuta para tener una visión resumida de las consultas y su estado. Fue seleccionada porque es la más: crítica, usada y se puede considerar como cuello de botella en el sistema. |
| **TRN02_1_FICHAS_DINAMICAS [INSERTAR_REGISTROS_ESPAÑOL]** | Esta transacción se ejecuta para ingresar datos en las fichas de los pacientes del Hospital Español. Fue seleccionada porque es crítica, de las más usadas y se puede considerar como cuello de botella en el sistema. |
| **TRN02_2_FICHAS_DINAMICAS [INSERTAR_REGISTROS_PEREIRA]** | Esta transacción se ejecuta para ingresar datos en las fichas de los pacientes del Hospital de la Mujer - CHPR Centro Hospitalario Pereira Rossell. Fue seleccionada porque es crítica, de las más usadas y se puede considerar como cuello de botella en el sistema. |
| **TRN03_HISTORIA_CLINICA [VISUALIZACION]** | Esta transacción se ejecuta para visualizar la historia clínica de los pacientes. Generar datos que figuren con datos patronímicos (como de admisión). Fue seleccionada porque es crítica, de las más usadas y se puede considerar como cuello de botella en el sistema. |
| **TRN04_ATENCION [FINALIZAR]** | Esta transacción se ejecuta para finalizar la atención (Ej.: terminó la consulta, el paciente va a "algún" lado). En general quedan pendientes (al menos hasta que no se agregue el otro grupo de transacciones). |
| **TRN05_BOX [ASIGNAR]** | Esta transacción se ejecuta para asignar un box a un paciente. |
| **TRN06_BOX [REASIGNAR]** | Esta transacción se ejecuta para desasignar un box asignado previamente a un paciente. |

La transacción dos tiene dos versiones pues las fichas de cada tipo de hospital son distintas y se constató que tuvieron tiempos de respuesta diferentes. Se definió una transacción con las fichas similares a las que se encuentran en el “Hospital Español” y otra transacción con la ficha que suele encontrarse en el “Hospital de la Mujer” (“Pereira Rossell”).

El detalle de los pasos se especificó de similar al primer caso (ver “Anexo VI – Guiones Salud”).

#### 5.2.2.2. Mezcla de Transacciones

El escenario a simular será un “peor caso” de lo comprendido durante las 24 horas de la operativa diaria de Noviembre y Diciembre de 2012. Se habla de un “peor caso” ya que por cada hora se tomaron los máximos usos de cada transacción a fin de simular la hora pico de mayor estrés detectado hasta el momento. Por lo tanto dicho escenario es el considerado más crítico y fue definido como el objetivo o “escenario del 100% de la carga”.

La siguiente tabla plantea la cantidad de usuarios esperados en el sistema en cada una de las transacciones. Para lograr en una hora la cantidad de transacciones deseada, se definen esperas entre las iteraciones de la operación para el mismo usuario virtual. Estas esperas entre usuarios se ajustan con los tiempos obtenidos de porcentajes menores de carga, partiendo para ello de los tiempos base y modelan las pausas que los usuarios realizan, en promedio, entre la ejecución de cada una de las iteraciones.

| Transacción | ~ # Usuarios | ~Iteraciones/Usuario /hora | ~Iteraciones/Hora |
|---|---|---|---|
| TRN01_VISOR_[CONSULTA_ORDENES_SERVICIO] | 214 | 33 | 7039 |
| TRN02_1_FICHAS_DINAMICAS_[INSERTAR_REGISTROS_ESPAÑOL] | 270 | 5 | 1350 |
| TRN02_2_FICHAS_DINAMICAS_[INSERTAR_REGISTROS_PEREIRA] | 133 | 2 | 300 |
| TRN03_HISTORIA_CLINICA_[VISUALIZACION] | 441 | 2 | 696 |
| TRN04_ATENCION_[FINALIZAR] | 214 | 7 | 1466 |
| TRN05_BOX_[ASIGNAR] | 100 | 13 | 1304 |
| TRN06_BOX_[REASIGNAR] | 44 | 31 | 1370 |

El escenario surge de realizar un estudio de los dos hospitales "tipo" que se encuentran en producción ("Pereira Rossell" y "Español") al momento de realizar esta prueba. Se categorizaron el resto de los hospitales del país en estos hospitales "tipo" considerando la cantidad de órdenes diarias que otorgan. Aquellos que otorgaban menos de 75 se categorizaron como "Español" mientras que los que superaban este número se como "Pereira".

#### 5.2.2.3. Criterio de aceptación

El criterio de aceptación definido fue que el percentil 90 de los tiempos de respuesta de cada uno de los pasos de las transacciones definidas fuera inferior a dos segundos.

### *5.2.3. Datos necesarios*

La base de datos utilizada en estas pruebas, contó con un volumen estimado al equivalente de seis meses de uso en producción. Esta base de datos se usó como punto de partida en cada ejecución de prueba y estaba poblada de la siguiente forma:

| Estado de las órdenes de servicio | Cantidad de registros |
|---|---:|
| Finalizadas | 828000 |
| Para finalizar | 8970 |
| Para asignar box | 8418 |
| Para reasignar box | 9576 |
| Para atender "Español" | 5306 |
| Para atender "Pereira" | 4836 |

## 5.3. Automatización y armado del ambiente de pruebas

### *5.3.1. Herramientas*

La herramienta seleccionada para automatizar las transacciones que utilizan el protocolo HTTP ha sido JMeter. Al tratase de una herramienta libre que se encuentra bajo licenciamiento GPL su utilización no tuvo costo adicional para el proyecto.

La utilización de esta herramienta ha sido posible debido a que el tráfico entre el cliente web y el servidor de aplicaciones es siempre HTTP. Las pruebas de concepto previo al inicio de las pruebas demostraron la factibilidad de la utilización de dicha herramienta siguiendo la metodología presentada.

### 5.3.2. Reproducción del escenario

Los escenarios se reprodujeron utilizando las capacidades que JMeter provee, oficiando de orquestador para los distintos programas de manera de simular correctamente todos los usuarios que hacen uso del sistema.
Cada uno de los scripts con JMeter se desarrolló de la siguiente manera:

- Grabación original del scripts.
- Validación de cada uno de los pasos del script. Esto implica la validación de cada una de las pantallas que tiene el sistema.
- Generalización de los scripts. Esto implica reemplazar constantes que se tienen en el momento de la grabación (usuarios, cuentas, entre otros) por variables a ser tomadas desde un archivo por cada una de las ejecuciones del script.
- Definición de la sección de login. Con esto se logró que cada una de las ejecuciones de los scripts por parte de los usuarios virtuales no realizaran el login. Esto imita lo que un usuario real realiza. Cada usuario realiza el login una única vez, manteniendo por lo tanto una única sesión en el sistema. Sin embrago, ante la falla de la transacción (la cual se da por datos erróneos o errores que se dieran en la aplicación), el usuario vuelve a realizar el *login*. Este comportamiento no aplica para la transacción 2_1 donde el usuario realiza el *login* y el *logout* en cada iteración.

Se definieron, en primera instancia, la cantidad de usuarios reales (ver sección "Escenario de carga") como usuarios virtuales. Cada uno de ellos ejecuta las transacciones asignadas. La cantidad de transacciones que cada uno de ellos ejecutó fue fijada a partir de datos referenciados por distintos actores que conocen del negocio. Se configuraron *delays* y *ramp-ups*.

Todas las pruebas realizadas cuentan con monitorización del sistema a fin de observar el comportamiento de los componentes involucrados y medir su rendimiento. En los casos en que se detectan patrones a analizar, se aumentan los niveles de *log* y monitores. Esta monitorización se describe a continuación.

## 5.4. Monitorización

Durante el despliegue del "Escenario de Infraestructura" definido para la simulación (es decir el "Armado del ambiente de pruebas") también se configuraron los monitores de primer nivel que recolectaron datos para detectar posibles cuellos de botella.

### 5.4.1. Indicadores

Cada una de las distintas capas de la arquitectura fue monitorizada para el posterior análisis de su desempeño y comportamiento ante las cargas impuestas en las pruebas. Tanto el servidor Web, como el servidor de base de datos fueron estudiados, utilizando para ello varios parámetros que se describen a continuación.

- **Servidores - Sistema Operativo (tanto de servidores como de generadoras)**: uso de CPU, uso de memoria, uso de la red, uso de IO.
- **Manejador de base de datos**: Slow Query log.
- **Servidor de aplicación**: Uso de memoria y tiempos de *Garbage Collection* (*heap de JVM*).
- **Generadoras**: Uso de memoria y tiempos de *Garbage Collection* (*heap* de *JVM*).

En algunas pruebas, debido a la necesidad de obtener mayores datos respecto al comportamiento de componentes de infraestructura y sistema, se utilizaron monitores que pueden ser considerados "intrusivos" (debido a su influencia en la performance del sistema) como ser JConsole (para el acceso a los datos JMX de Tomcat) y MySQLReport de MySQL.

### *5.4.2. Herramientas utilizadas*

Las herramientas utilizadas para la monitorización de los distintos componentes fueron:

- **nmon:** para los recursos del el SUT.
- **Perfmon:** para los sistemas operativos de las generadoras de carga
- **GC Viewer**: para observar el comportamiento de la pila (*heap*) de memoria de los servidores de aplicación y de las generadoras de carga.

## 5.5. Ejecución de pruebas

Cómo plantea la metodología no se ataca al sistema directamente con el 100% de la carga esperada, sino que se adopta un esquema de ejecuciones incrementales, comenzando primero por la ejecución de las líneas base, las cuales simulan la interacción de un usuario con el sistema, y son considerados por lo tanto como los tiempos en el mejor caso. Luego se decidió ejecutar con un 20% de la carga e incrementando, dependiendo de los resultados obtenidos.

Esta metodología de pruebas incrementales permite encontrar los problemas más severos en etapas tempranas y luego ir refinando a medida que se va avanzando en el proyecto.

En esta sección se describen las pruebas realizadas y los resultados encontrados en las mismas. Para obtener todos los detalles de las pruebas realizadas, se recomienda consultar los informes de ejecución de cada una de ellas.

### *5.5.1. Cronograma de las pruebas*

En el cronograma original, estaba planteado realizar las pruebas durante la sexta y séptima semana del proyecto con una duración de 10 días. Debido a que los resultados obtenidos no eran considerados aceptables en diferentes escenarios ejecutados, se extendió este plazo realizándose entonces, ejecuciones lo largo de ocho semanas aproximadamente (lo cuál considera pausas intermedias con el objetivo de generar nuevas versiones con cambios identificados en las pruebas anteriores).

La primera ejecución de los tiempos base se realizó el viernes 28 de diciembre de 2012, y dado los problemas que presentaba el SUT, se continuó variando y ejecutando pruebas similares en cuatro escenarios diferentes de infraestructuras hasta el día viernes 11 de enero de 2013, cuando se decidió detener la ejecución de pruebas con el objetivo de que se pueda trabajar en mejorar la aplicación. El lunes 28 de enero de 2013, se continuó con las ejecuciones, primero de los tiempos base. El 4 de febrero de 2013, se ejecutó la primer prueba del 20%, y luego se fue incrementando la carga gradualmente. A medida que se ejecutaban las pruebas se fue cambiando la aplicación con el objetivo de mejorar los tiempos de respuesta que se obtenían.

Los escenarios de infraestructura consistieron en máquinas virtuales (VM):

1. Servidor de aplicaciones y DBMS en una VMWare.
2. Servidor de aplicaciones y DBMS en dos VMWare, en un *host*.
3. Servidor de aplicaciones y DBMS en dos VMWare, en dos *hosts*.
4. Servidor de aplicaciones y DBMS en dos Virtual Box, en dos *hosts*.

La última sería la infraestructura equivalente a la de producción.

A partir del segundo mes, cuando los tiempos de los tiempos base fueron aceptables, las pruebas se centraron en la arquitectura con servidor de aplicaciones y servidor de base de datos en hosts separados cada uno en una máquina virtual (VMWare) que se encontraba en Geocom. Debido a que los problemas identificados se encontraban en la aplicación y los mismos podían ser

identificados de manera independiente a la plataforma en la que se ejecutaba, se decidió trabajar en un ambiente cerca de los desarrolladores como fue el descripto anteriormente.

Las pruebas descriptas en este caso son aquellos que permitieron avanzar en el proyecto. Existieron otras ejecuciones que no son reportadas debido al poco aporte de las mismas o a que su finalidad era la de probar ajustes puntuales que no aportaron mejores resultados.

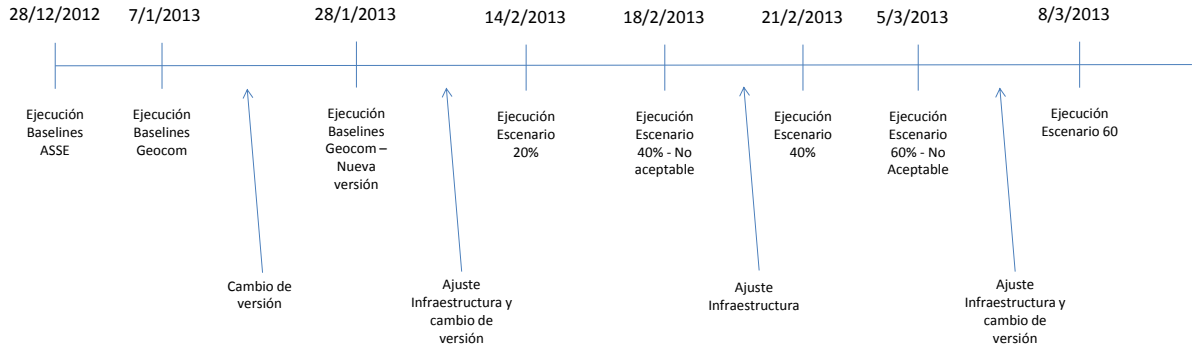



**Figura 16: Calendario de las principales ejecuciones y cambios**

## *5.5.2. Resultados*

A continuación, se mostrarán los resultados más relevantes de las pruebas realizadas con el sistema Geosalud.

### 5.5.2.1. Tiempos Base

Fue necesario ejecutar varias veces los tiempos base. Las primeras ejecuciones daban tiempos inaceptables para la operativa, el "Anexo VII – Baselines Salud" presenta las distintas ejecuciones y cambios realizados hasta alcanzar tiempos que satisficieron las expectativas.

El primer incremento de carga fue el 20% y los sucesivos incrementos fueron del orden del 20%.

### 5.5.2.2. 20 % de Carga

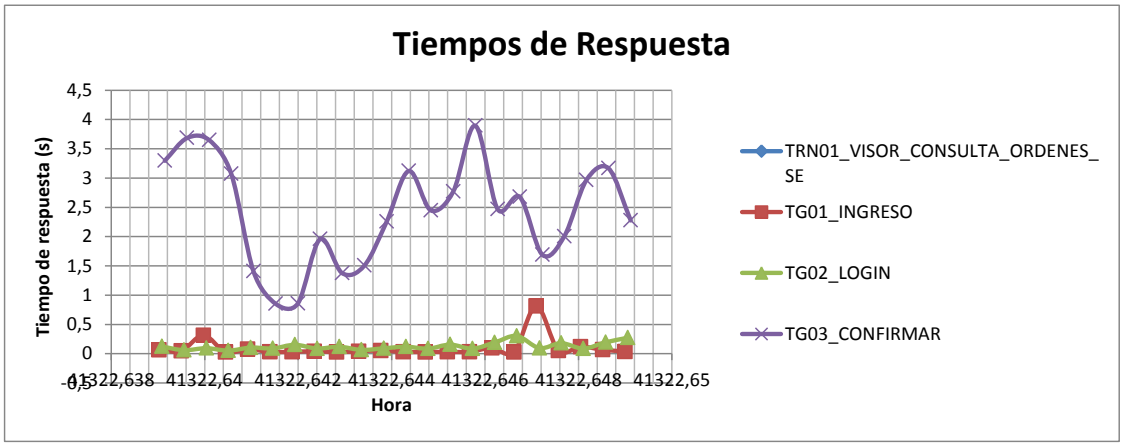
En el escenario de 20% el sistema es ejecutado por 283 usuarios ejecutando un aproximado de 2900 transacciones de negocio en una hora según la distribución que puede observarse en la siguiente tabla.

| Transacción | Iteraciones/Usuario /hora | Usuarios según carga |
|---|---|---|
| TRN01_VISOR_[CONSULTA_ORDENES_SERVICIO] | 33 | 43 |
| TRN02_1_FICHAS_DINAMICAS_[INSERTAR_REGISTROS_ESPAÑOL] | 5 | 54 |
| TRN02_2_FICHAS_DINAMICAS_[INSERTAR_REGISTROS_PEREIRA] | 2 | 27 |
| TRN03_HISTORIA_CLINICA_[VISUALIZACION] | 3 | 88 |
| TRN04_ATENCION_[FINALIZAR] | 7 | 43 |
| TRN05_BOX_[ASIGNAR] | 13 | 20 |
| TRN06_BOX_[REASIGNAR] | 31 | 9 |

Los usuarios se integraron a la prueba según un *ramp-up* de 30 minutos. Esto define la manera en la que entran los usuarios a la aplicación y, por ende, se tienen los momentos de mayor carga.

### TRN01_VISOR_[CONSULTA_ORDENES_SERVICIO]

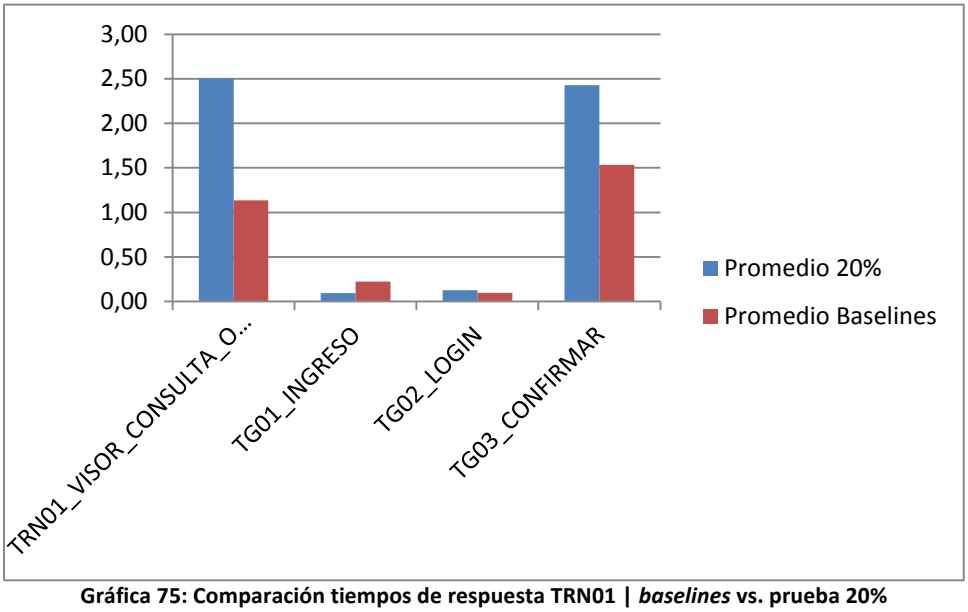
Los tiempos obtenidos durante la ejecución de las pruebas pueden verse en la gráfica que se encuentra a continuación. En el mismo puede verse cada uno de los tiempos de los pasos de la transacción.




**Gráfica 74: Tiempos de respuesta para la TRN01 | prueba 20%**

Como se ve en el análisis de la ejecución de los tiempos base, los pasos que determinan el tiempo total de la transacción se relacionan de manera directa con el visor. Ya sea la visualización de las órdenes o el paso de confirmación, ambos despliegan el visor de órdenes y son las que consumen mayor tiempo. Estos tiempos van desde un mínimo de menos de un segundo a un máximo de cuatro segundos, lo cual indica que la concurrencia o la combinación usuario / hospital influye sobre dicha transacción.

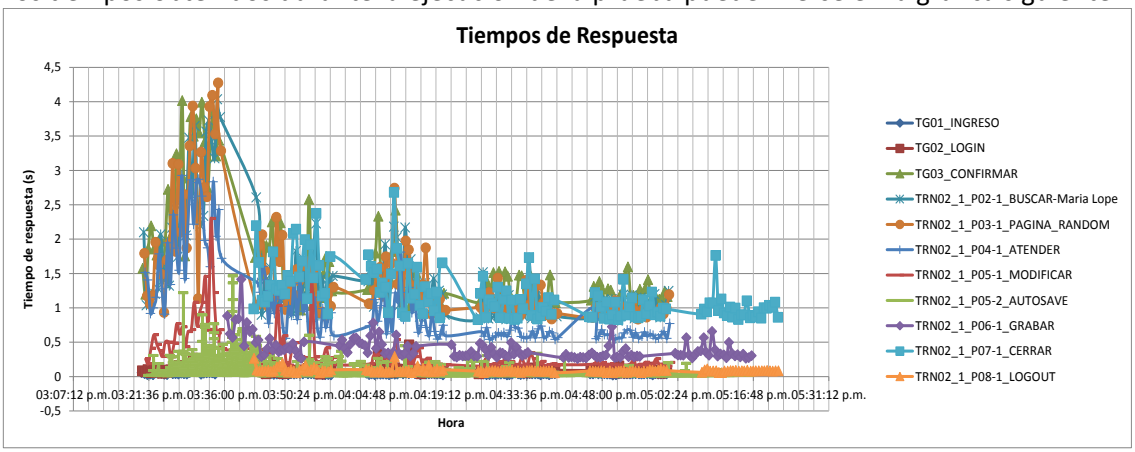
En cuanto a la comparación con los tiempos base, el promedio se ve acrecentado, principalmente por los tiempos máximos obtenidos.




**Gráfica 75: Comparación tiempos de respuesta TRN01 | *baselines* vs. prueba 20%**

### *TRN02_1_FICHAS_DINAMICAS_[INSERTAR_REGISTROS_ESPAÑOL]*

Los tiempos obtenidos durante la ejecución de la prueba pueden verse en la gráfica siguiente

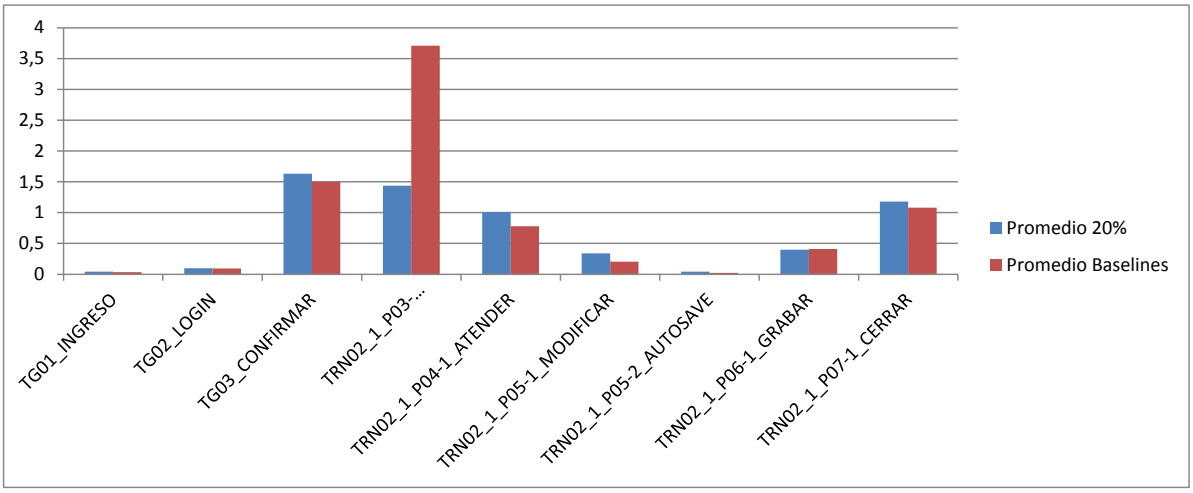



**Gráfica 76: Tiempos de respuesta para la TRN02_1 | prueba 20%**

Todas las transacciones evidencian un pico en los tiempos durante el período de entrada de los usuarios al sistema. Esta transacción tiene la particularidad que los usuarios realizan el *login* y el *logout* en cada acción (imitando lo que sucede generalmente en el "Hospital Español").

Las transacciones relacionadas con el visor, nuevamente, son las que muestran mayor variación de tiempos. Algunos pasos, como el que se encarga de guardar la ficha, prácticamente no varían los tiempos de respuesta y registran valores en el entorno del segundo de ejecución.

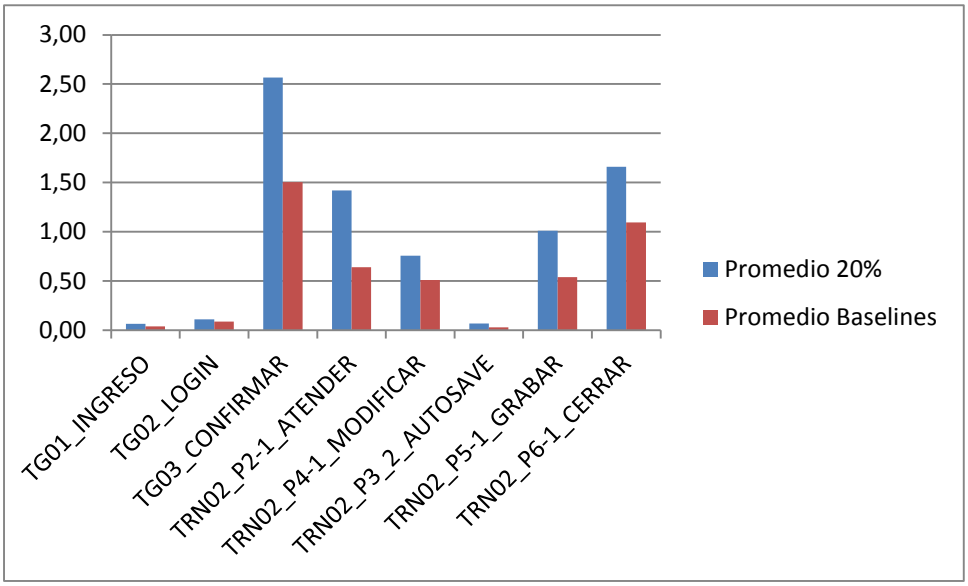
La comparación contra los tiempos base (en su promedio) pueden verse en la siguiente gráfica.




**Gráfica 77: Comparación tiempos de respuesta TRN02_1 | *baselines* vs. prueba 20%**

### TRN02_2_FICHAS_DINAMICAS_[INSERTAR_REGISTROS_PEREIRA]

Se detectó un comportamiento distinto respecto a la transacción anterior y se da en el paso de grabación. En el caso de las fichas del "Hospital Español", los tiempos no se veían influenciados por la concurrencia mayor que se da en la etapa de *ramp-up*, mientras que en las fichas tipo "Pereira Rossell" si lo hace. De todas maneras, los tiempos nunca son mayores a dos segundos. La comparación con los tiempos de base evidencia la diferencia, ya que comparando el paso de grabación en ambos escenarios, el decremento de performance es mayor en esta transacción.

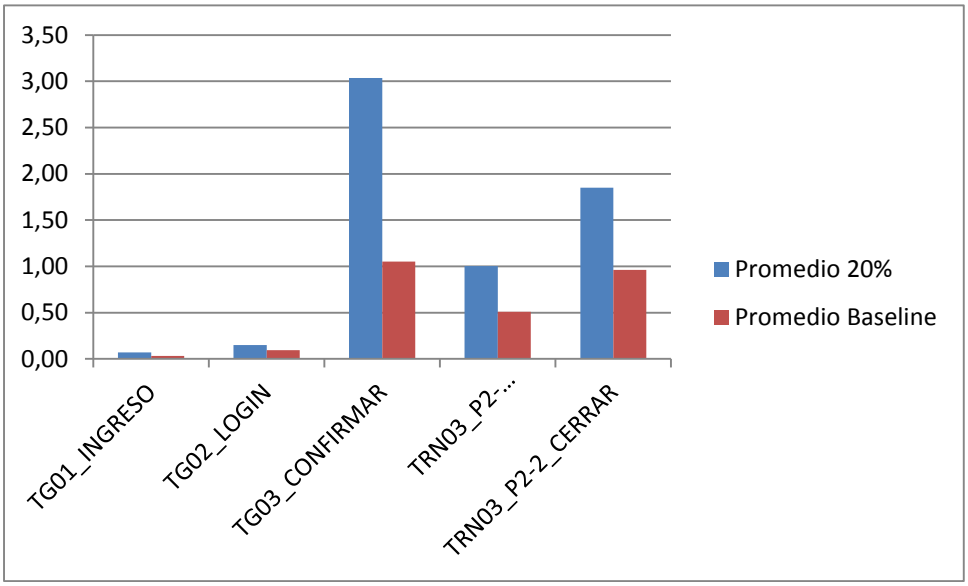



**Gráfica 78: Comparación tiempos de respuesta TRN02_2 | *baselines* vs. prueba 20%**

### TRN03_HISTORIA_CLINICA_[VISUALIZACION]

En el período de *ramp-up* los tiempos de respuesta relacionados con el visor de órdenes y los pasos que terminan en dicha pantalla aumentan hasta cinco segundos.

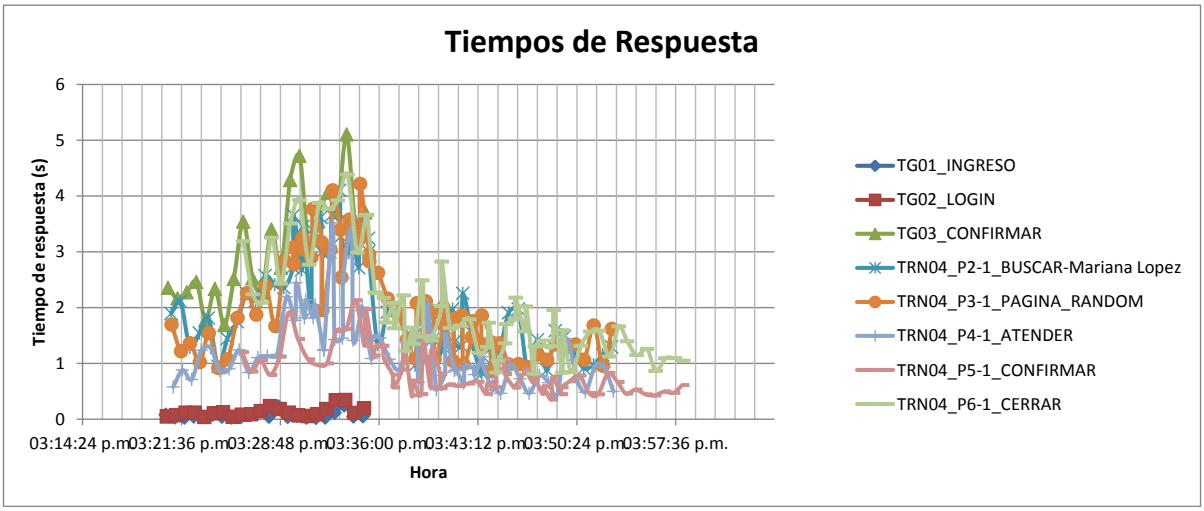
La comparación contra los tiempos de base se pueden observar en la siguiente gráfica. Nuevamente, los promedios se ven fuertemente influenciados por los tiempos de las transacciones que se ejecutaron durante el tiempo de *ramp-up*.




**Gráfica 79: Comparación tiempos de respuesta TRN03 | *baselines* vs. prueba 20%**

### TRN04_ATENCION_[FINALIZAR]

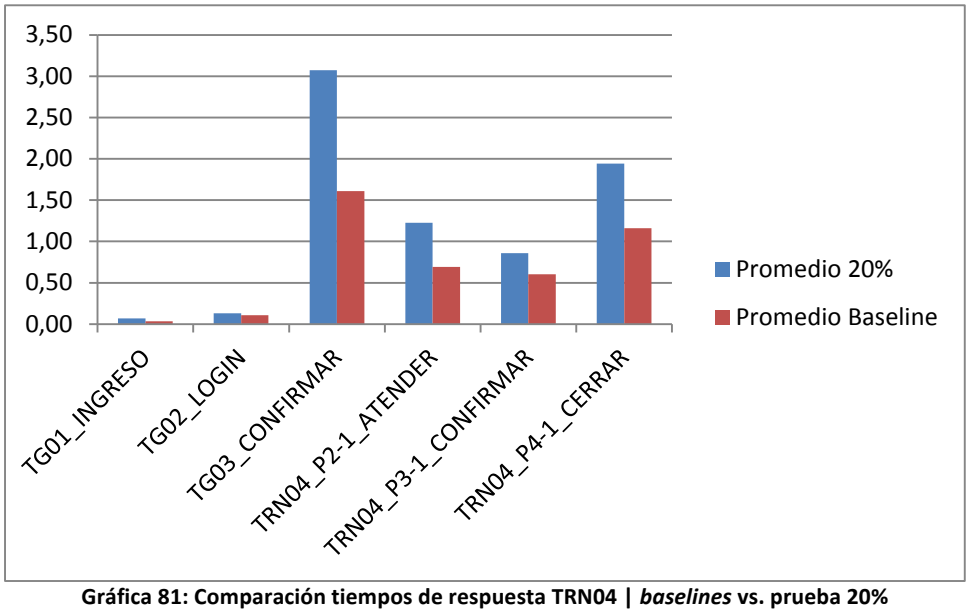
Los tiempos obtenidos para la transacción de finalización de atención pueden observarse en la gráfica que se encuentra a continuación.




**Gráfica 80: Tiempos de respuesta para la TRN04 | prueba 20%**

El patrón de la gráfica es similar al del resto de las transacciones, existiendo picos de tiempos durante el *ramp-up* de los usuarios. Luego de este período de *ramp-up* existen variaciones, pero en ninguno de los casos se encuentra por arriba de los dos segundos.

La comparación con los tiempos de base pueden verse en la siguiente gráfica.

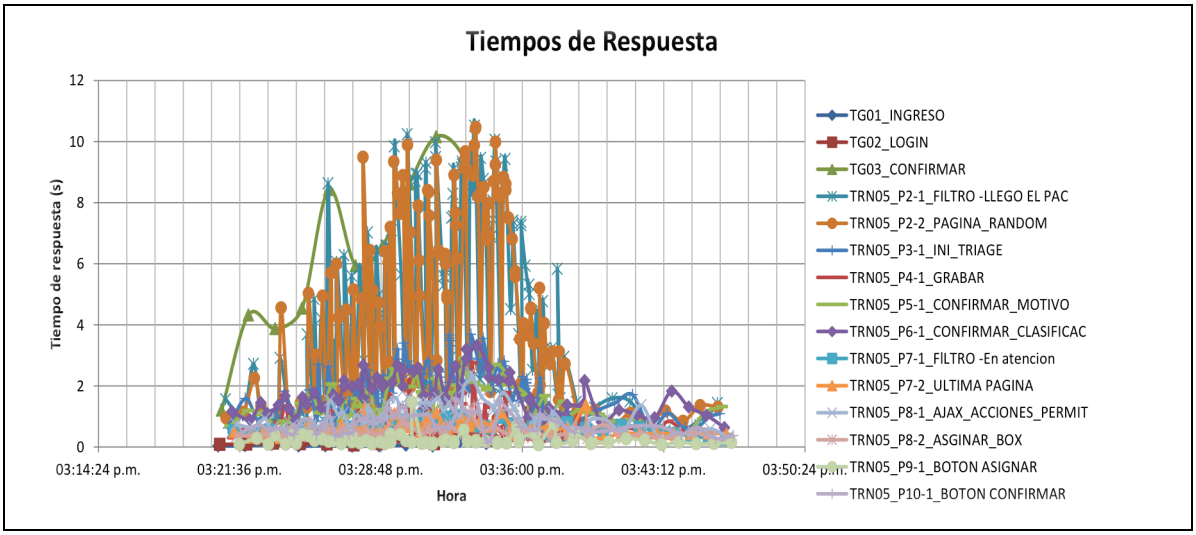



**Gráfica 81: Comparación tiempos de respuesta TRN04 | *baselines* vs. prueba 20%**

Los aumentos más significativos se vuelven a dar en los pasos que finalizan en el visor de órdenes.

### TRN05_BOX_[ASIGNAR]

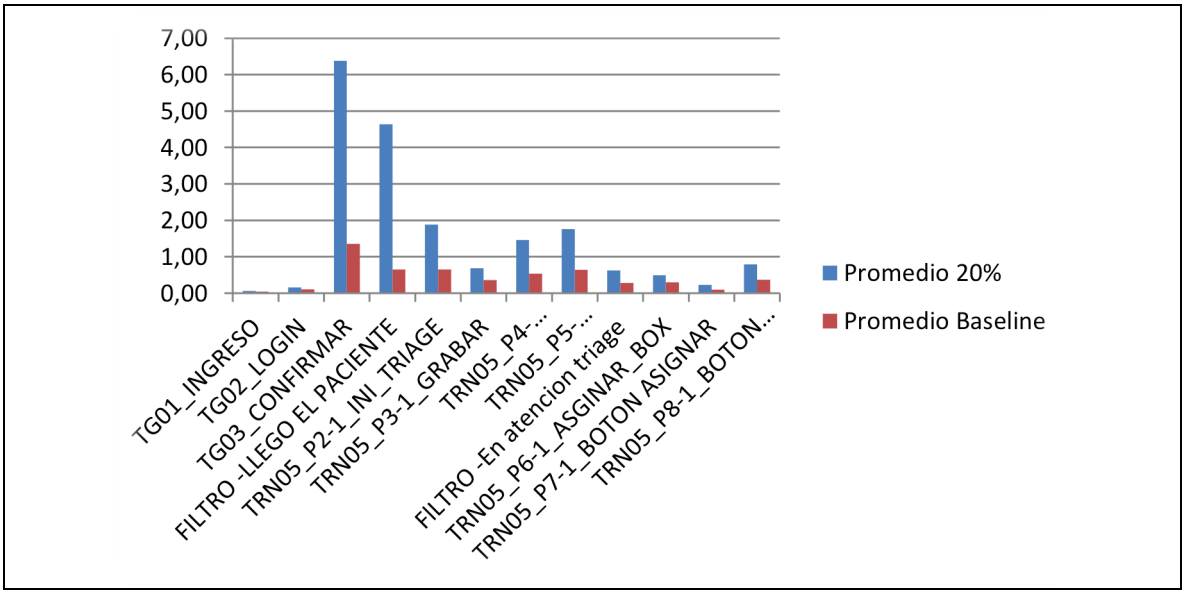
Los tiempos obtenidos para esta transacción pueden verse en la gráfica siguiente.




**Gráfica 82: Tiempos de respuesta para la TRN05 | prueba 20%**

En esta transacción puede verse una variación muy importante en los tiempos de los pasos relativos con la visualización de órdenes. Estos se dan incluso por fuera del tiempo de ramp-up del escenario. Es importante hacer notar que los usuarios que utilizan esta transacción son diferentes a los usuarios que utilizan las transacciones anteriores (en este caso, se trata de "triagistas"). Verificando las ejecuciones de manera manual (sin carga), se puede ver que usuarios diferentes tienen tiempos de respuesta distintos, por lo que si bien los tiempos pueden verse influenciados por la concurrencia, existe un factor dependiente fuertemente de los usuarios y los datos con los que se ejecuta la transacción.

Esta variación importante entre los tiempos máximos y mínimos de ejecución de las transacciones hace que los promedios respecto a los obtenidos en los tiempos de base sean mucho mayores, en algunos casos multiplicando por seis los mismos.

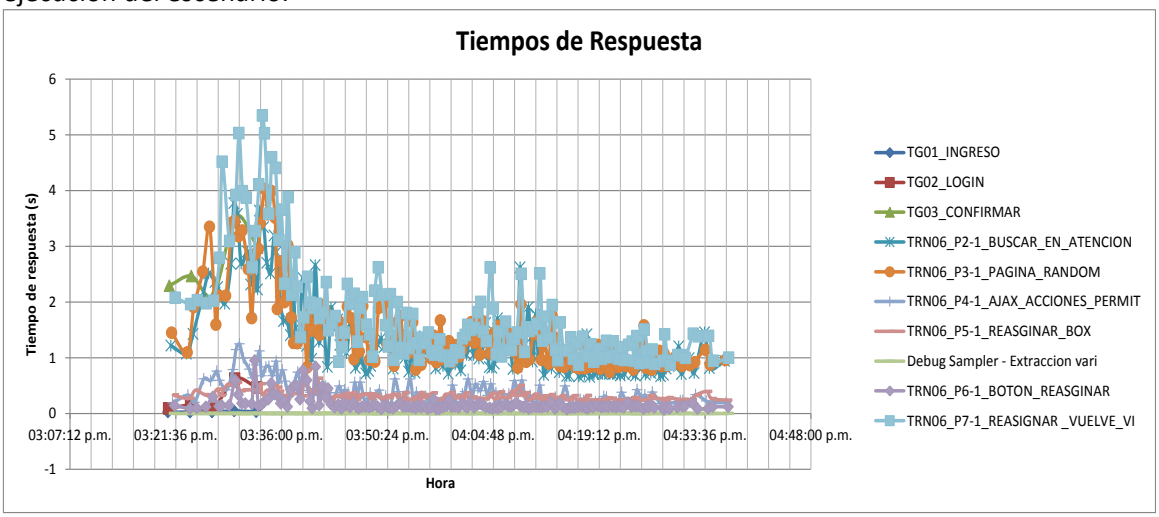



**Gráfica 83: Comparación tiempos de respuesta TRN05 | *baselines* vs. prueba 20%**

Se recomienda la revisión de este comportamiento (el cuál como se dijo anteriormente, puede ser obtenido sin concurrencia) con el fin de mejorar los tiempos que se obtienen con algunos de los datos de entrada en el caso de los usuarios de tipo “triagistas”.

### TRN06_BOX_[REASIGNAR]

A continuación se muestran los tiempos de respuesta obtenidos para la transacción durante la ejecución del escenario.




**Gráfica 84: Tiempos de respuesta para la TRN06 | prueba 20%**

En esta transacción vuelven a verse los patrones relacionados con los tiempos mayores durante el *ramp-up* de los usuarios, en especial en los pasos que se encuentran relacionados con la visualización de las órdenes.

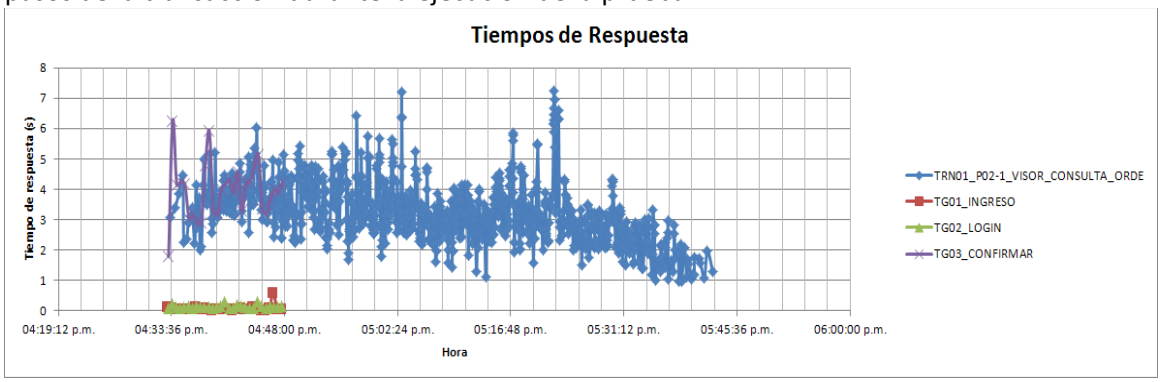
#### 5.5.2.3. 40% de la Carga

Siguiendo el mismo razonamiento que en la prueba del 20%, se decidió luego del resultado de las pruebas anteriores, realizar la ejecución del escenario del 40% de la carga. La prueba aquí descrita (realizada el 21 de Febrero) tuvo un antecedente de ejecución no satisfactorio que se realizó el 18 de Febrero. Luego de la primera prueba, se realizaron los siguientes cambios a la aplicación e infraestructura:

- Se agregan 2 cores (quedando en 4), 2Gb de RAM (quedando en 4Gb), se aumenta el espacio en disco (quedando en 21Gb y 11Gb libres).
- Se aplican *fixes*
    - Mejoras del visor y el filtro "*Llega Paciente*".
    - Cambios de consultas SQL, agregando ORs en las sentencias del visor.

### TRN01_VISOR_[CONSULTA_ORDENES_SERVICIO]

En la siguiente gráfica pueden verse los tiempos de respuesta obtenidos para cada uno de los pasos de la transacción durante la ejecución de la prueba.



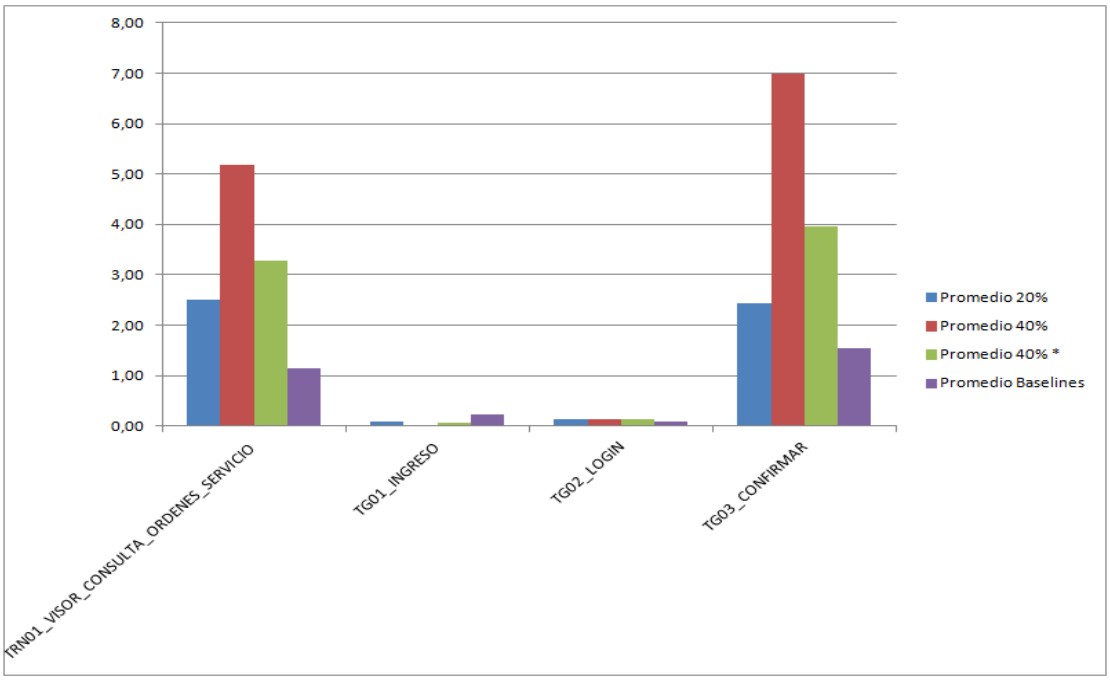

**Gráfica 85: Tiempos de respuesta para la TRN01 | prueba 40%**

Como pudieron verse, los pasos que determinan el tiempo total de la transacción se relacionan de manera directa con el visor. Ya sea la visualización de las órdenes o el paso de confirmación, ambos despliegan el visor de órdenes y son las que consumen mayor tiempo. Estos tiempos van desde un mínimo de menos de dos segundos a un máximo de siete segundos, lo cual indica que la concurrencia o la combinación usuario / hospital influye sobre dicha transacción.

En cuanto a la comparación con los tiempos base, el promedio se puede ver en la gráfica verde (abajo - Promedio 40%*) el cual sigue acrecentado, pero se observa la mejora respecto al 40% con menos recursos.

El promedio total del paso de visualización de las órdenes provenientes de una confirmación se ubica en los cuatro segundos, lo cual es casi dos veces el tiempo que se da en la ejecución del 20% nuevamente se ven mejoras respecto al 40% con menos recursos.

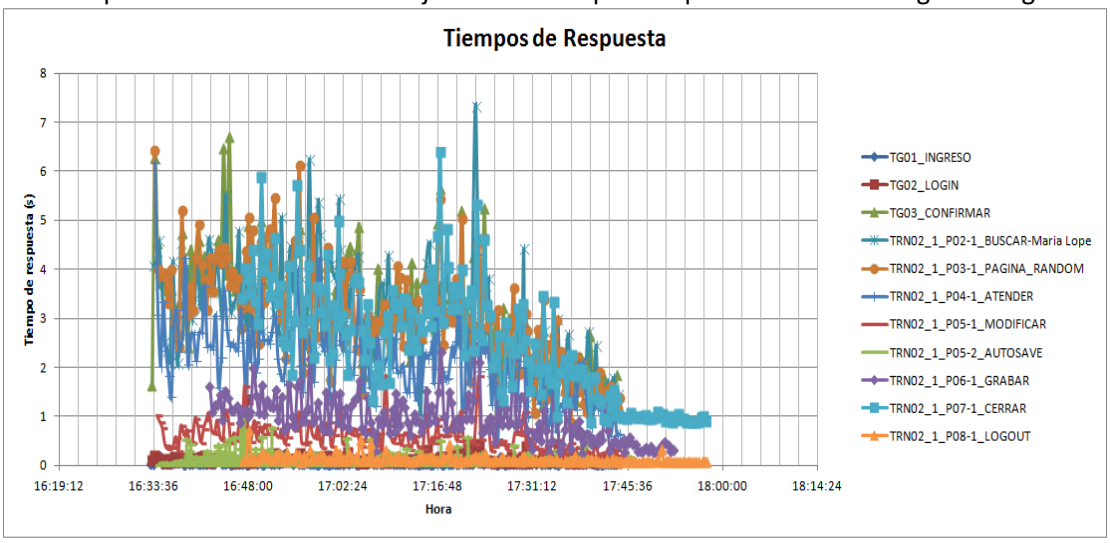
La comparación de los tiempos promedio de los diferentes escenarios ejecutados puede verse en la siguiente gráfica.




**Gráfica 86: Comparación tiempos de respuesta TRN01 | prueba 40%**

## TRN02_1_FICHAS_DINAMICAS_[INSERTAR_REGISTROS_ESPAÑOL]

Los tiempos obtenidos durante la ejecución de la prueba pueden verse en la gráfica siguiente

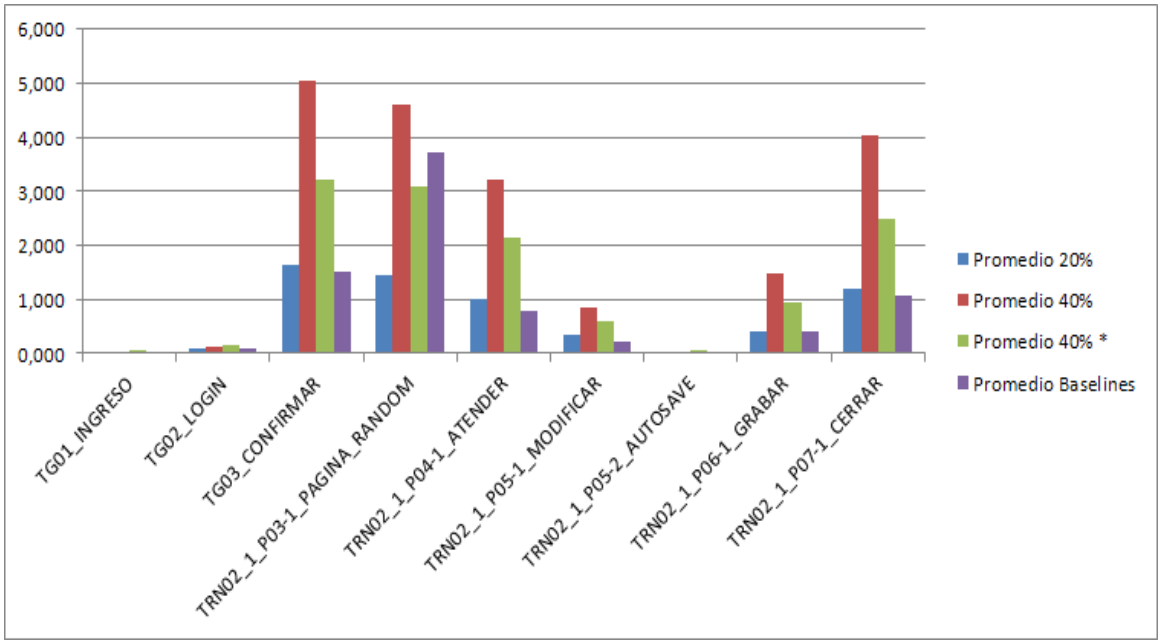



**Gráfica 87: Tiempos de respuesta para la TRN02_1 | prueba 40%**

En la gráfica se puede ver el mismo patrón de comportamiento en cuanto a los tiempos asociados al visor. Los tiempos de respuesta oscilan entre uno y siete segundos. Los mejores tiempos se observan sobre el final de la prueba donde hay menos carga.

En los peores tiempos se tienen respuestas que rondan los siete segundos lo cual muestra una mejora respecto a la prueba anterior pero siguen siendo altos.

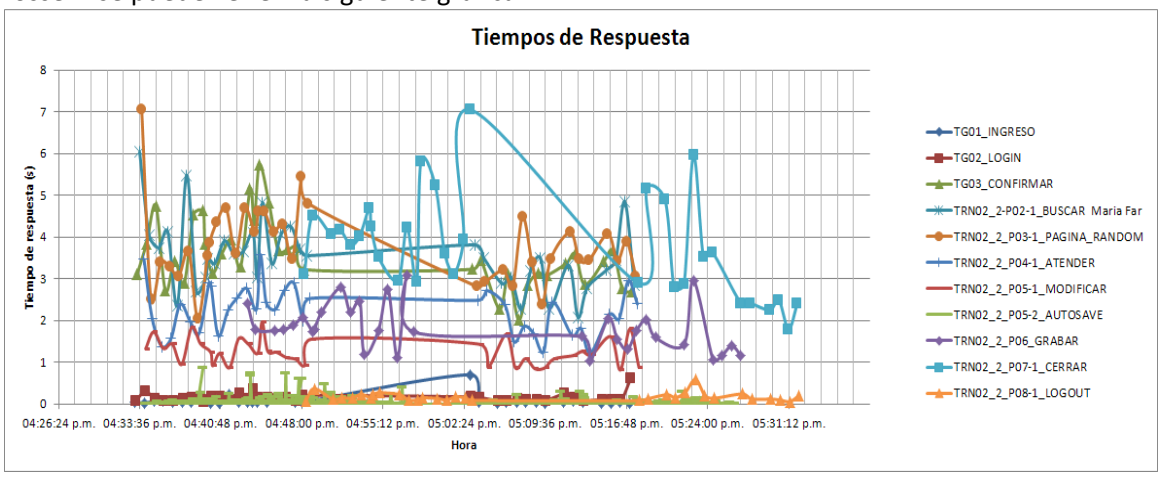
La comparación contra los tiempos base (en su promedio) pueden verse en la siguiente gráfica. Se ve una mejora respecto a la prueba con menos recursos pero no lo suficiente para alcanzar los tiempos obtenidos en el 20%.




**Gráfica 88: Comparación tiempos de respuesta TRN02_1 | prueba 40%**

### TRN02_2_FICHAS_DINAMICAS_[INSERTAR_REGISTROS_PEREIRA]

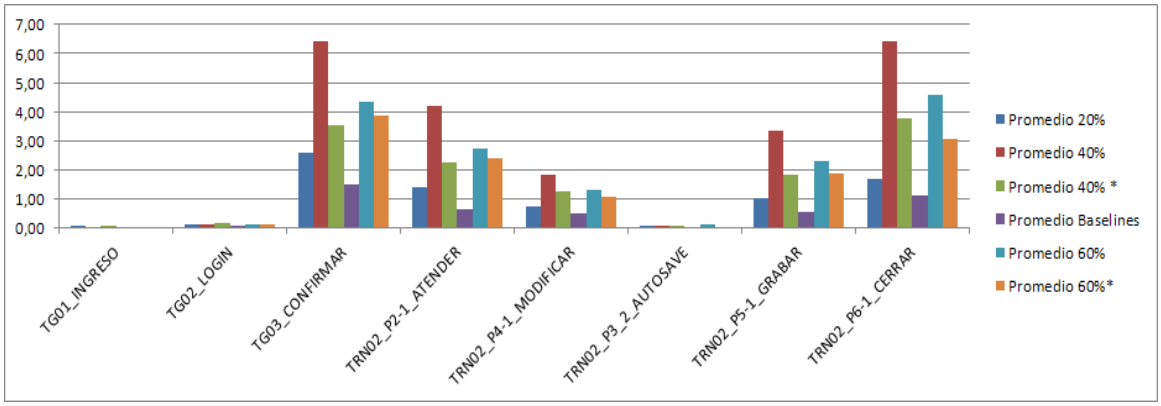
Los tiempos que se dieron en la transacción de grabación de fichas dinámicas del tipo “Pereira Rossell” se puede ver en la siguiente gráfica.




**Gráfica 89: Tiempos de respuesta para la TRN02_2 | prueba 40%**

El mismo comportamiento se repite en los tiempos de esta transacción, una observación es que los tiempos de respuesta de las transacciones que finalizan en el visor son bastante parejos a lo largo de toda la prueba lo que habla de que el cambio del 20% al 40% (dejando de lado el cuello de botella por recursos) es importante, deja de notarse el impacto de los *logins* para estar todo el lapso con tiempos un poco más altos.

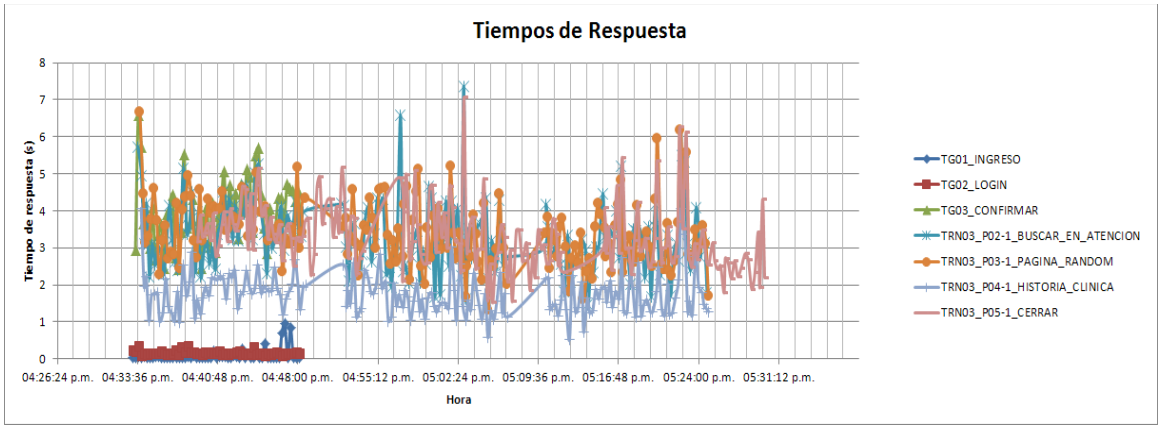
En la comparación con los tiempos de los demás escenarios se puede notar la mejora respecto a la prueba del 60% y se podría decir que prácticamente se equipara con la prueba del 40%*.




**Gráfica 90: Comparación tiempos de respuesta TRN02_2 | prueba 40%**

### TRN03_HISTORIA_CLINICA_[VISUALIZACION]

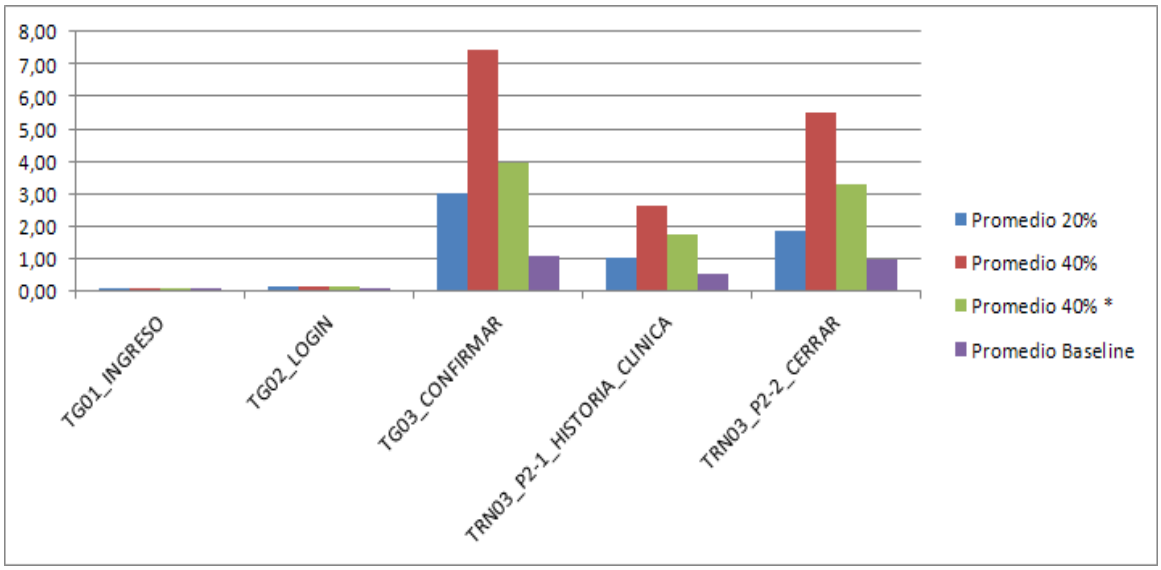
Los tiempos obtenidos para los diferentes pasos de transacción se pueden ver a continuación.




**Gráfica 91: Comparación tiempos de respuesta TRN03 | prueba 40%**

Nuevamente se observa el mismo comportamiento y de manera coincidente con las otras transacciones. Esto confirma el comportamiento observado anteriormente.

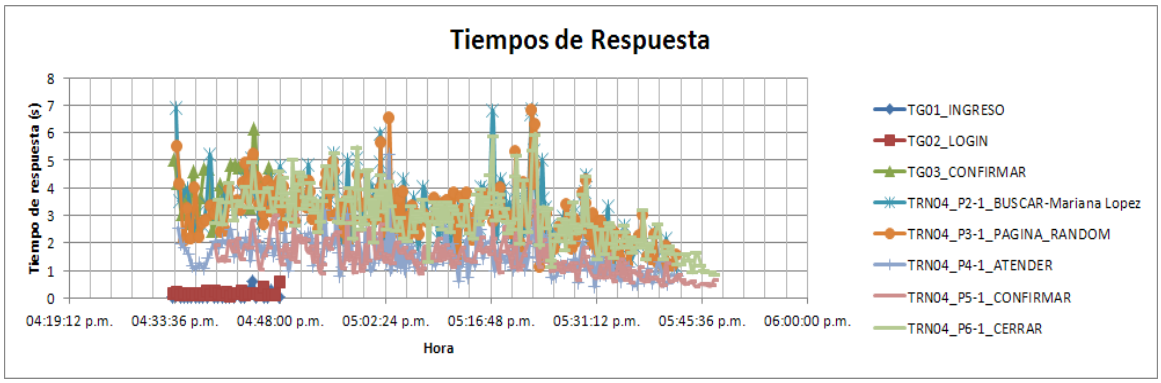
La comparación contra demás escenarios en cuanto al promedio arroja conclusiones similares a las transacciones anteriores.




**Gráfica 92: Comparación tiempos de respuesta TRN03 | prueba 40%**

### TRN04_ATENCION_[FINALIZAR]

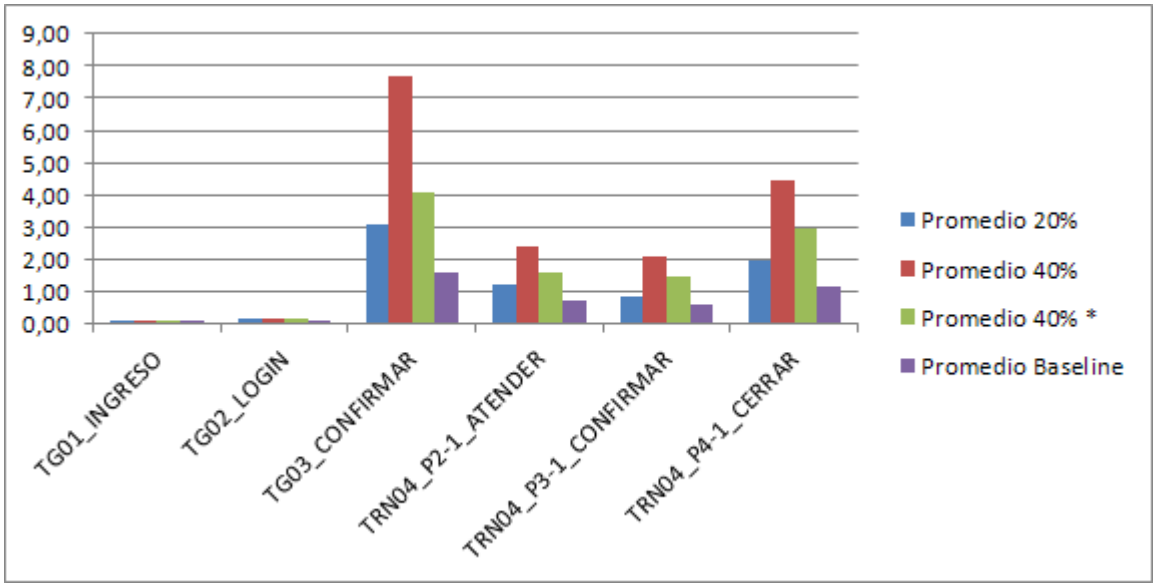
Los tiempos obtenidos para la transacción de finalización de atención pueden observarse en la gráfica que se encuentra a continuación.




**Gráfica 93: Tiempos de respuesta para la TRN04 | prueba 40%**

El patrón de la gráfica es similar al del resto de las transacciones, llegando los tiempos de respuesta a valores similares.

La comparación con los tiempos de los escenarios ya ejecutados pueden verse a continuación.

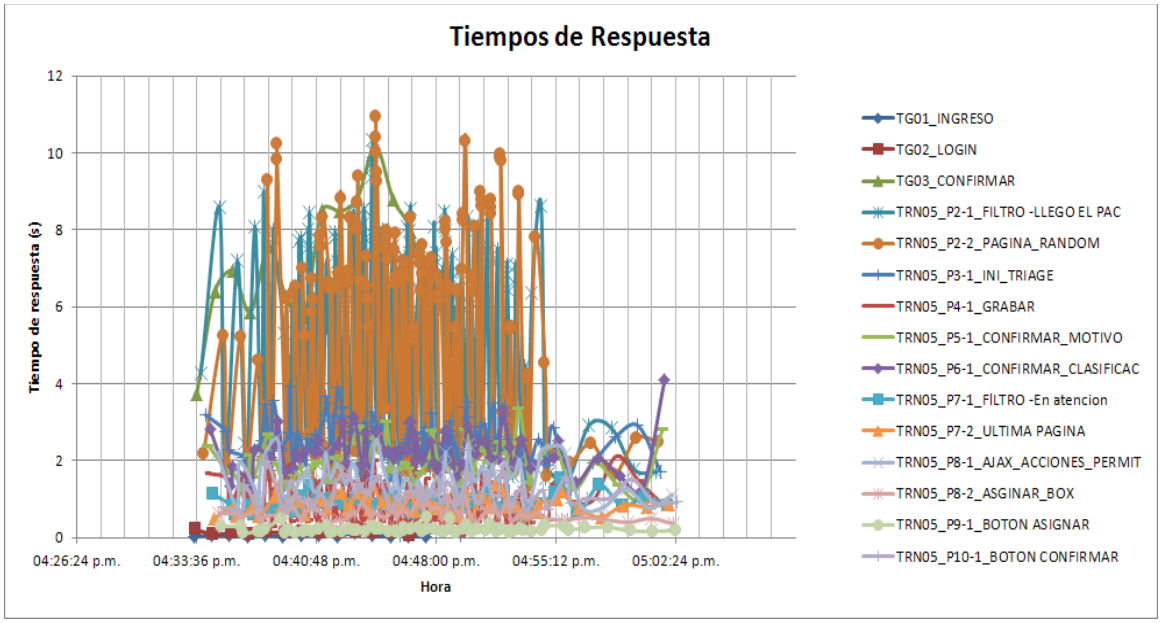



**Gráfica 94: Comparación tiempos de respuesta TRN04 | prueba 40%**

Los aumentos más significativos se vuelven a dar en los pasos que finalizan en el visor de órdenes.

### TRN05_BOX_[ASIGNAR]

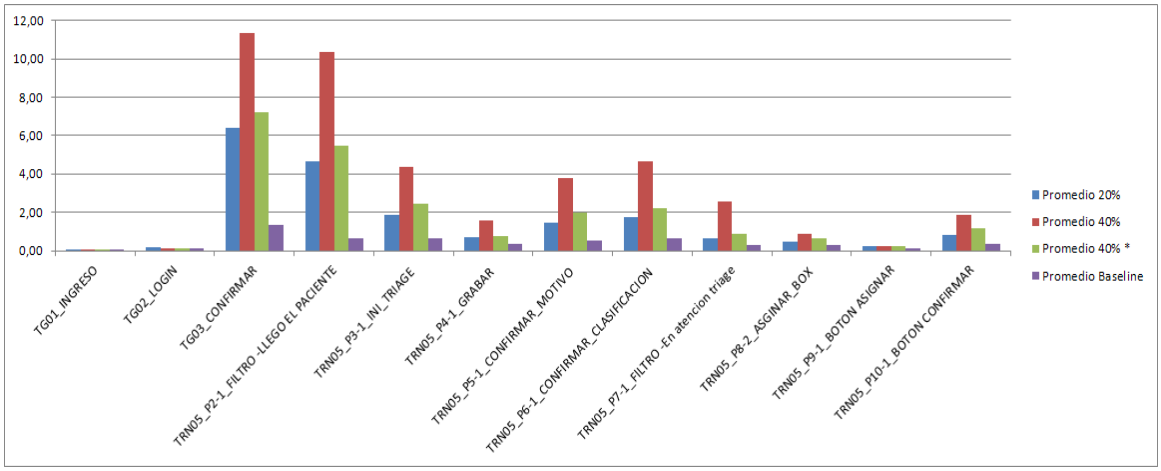
Los tiempos obtenidos para esta transacción pueden verse en la gráfica siguiente.




**Gráfica 95: Tiempos de respuesta para la TRN05 | prueba 40%**

En el caso de la asignación de boxes los tiempos superan el promedio. Y por lo tanto se repite el comportamiento que se dio en la ejecución del 20% respecto a la variabilidad según el usuario que ejecuta (que dio origen a la desactivación de la alarma de llegada de paciente), por lo que se recomienda seguir evaluando mejoras a estos pasos.

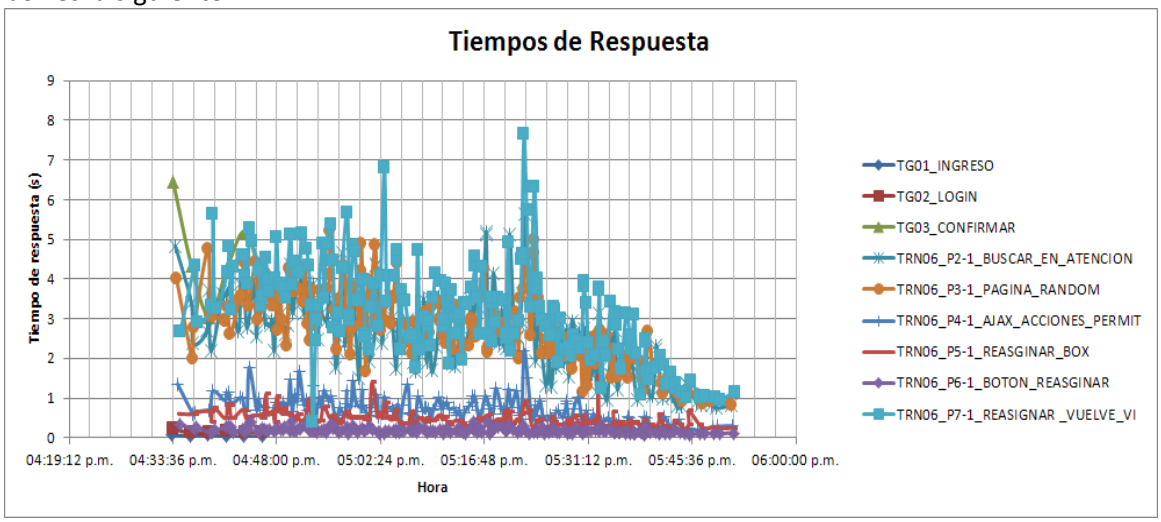
La comparación de los tiempos promedios de los distintos escenarios ejecutados para esta transacción puede verse en la siguiente gráfica.




**Gráfica 96: Comparación tiempos de respuesta TRN05 | prueba 40%**

### TRN06_BOX_[REASIGNAR]

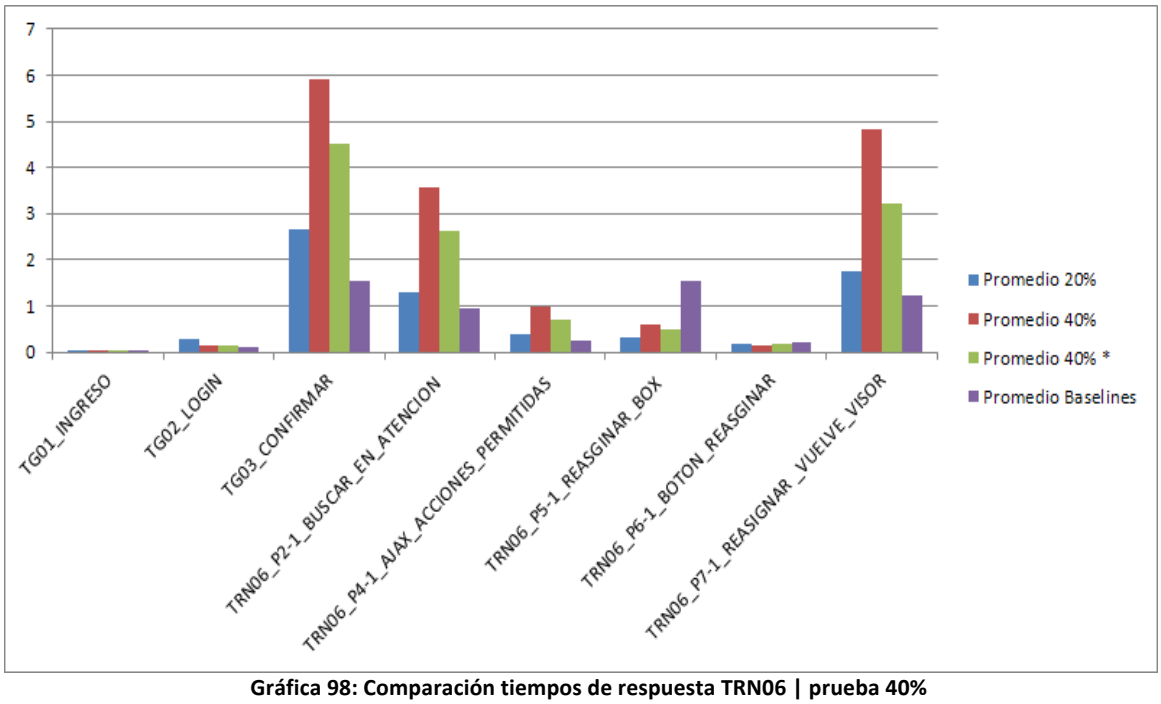
Los tiempos obtenidos durante la ejecución del escenario para la transacción de reasignación de box es la siguiente.




**Gráfica 97: Tiempos de respuesta para la TRN06 | prueba 40%**

En esta transacción vuelven a verse los patrones que se repiten en todas las transacciones.

Comparando los tiempos promedios obtenidos en el escenario del 40% contra los otros escenarios, se repite el mismo patrón que para las restantes transacciones.




**Gráfica 98: Comparación tiempos de respuesta TRN06 | prueba 40%**

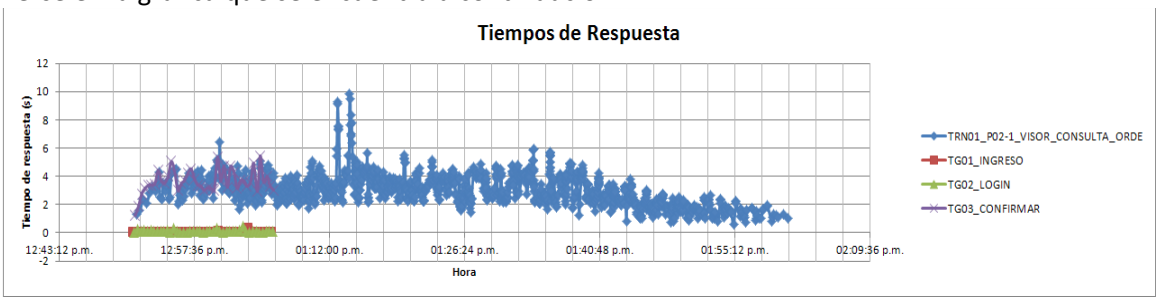
#### 5.5.2.4. 60% de la Carga

Una vez realizada la ejecución del escenario de carga del 40% y habiendo obtenido tiempos aceptables para el escenario planteado, se realizó la prueba del 60% de carga.
Dado que no se alcanzaban los criterios de aceptación definidos esta carga se ejecutó varias veces. Se realizaron los siguientes ajustes a la aplicación y se alcanzaron los criterios:

- Modificación de parámetros en la configuración de la base de datos MariaDB. A continuación se especifican los parámetros modificados su valor final.
    - innodb_buffer_pool_size=2147483648
    - innodb_additional_mem_pool_size=2097152
    - innodb_data_home_dir=
    - innodb_data_file_path=/dev/raw/raw1:10200raw
    - innodb_flush_log_at_trx_commit=2
    - innodb_flush_method=O_DIRECT
    - innodb_force_recovery=0
    - innodb_io_capacity=200
    - innodb_lock_wait_timeout=120
    - innodb_log_buffer_size=8388608
    - innodb_log_file_size=268435456
    - innodb_log_files_in_group=3
    - innodb_log_group_home_dir=/home/mysql
    - innodb_thread_concurrency=8
    - max_connections=500
    - wait_timeout=28800
- Se agregó un disco nuevo al servidor de base de datos sin formatear. El mismo es asignado en "crudo" o "raw" a mysql para almacenar los datos, de este modo la escritura y lectura la puede realizar sin pasar por el sistema operativo.

## TRN01_VISOR_[CONSULTA_ORDENES_SERVICIO]

Los tiempos de los pasos de la transacción obtenidos durante la ejecución de las pruebas pueden verse en la gráfica que se encuentra a continuación.



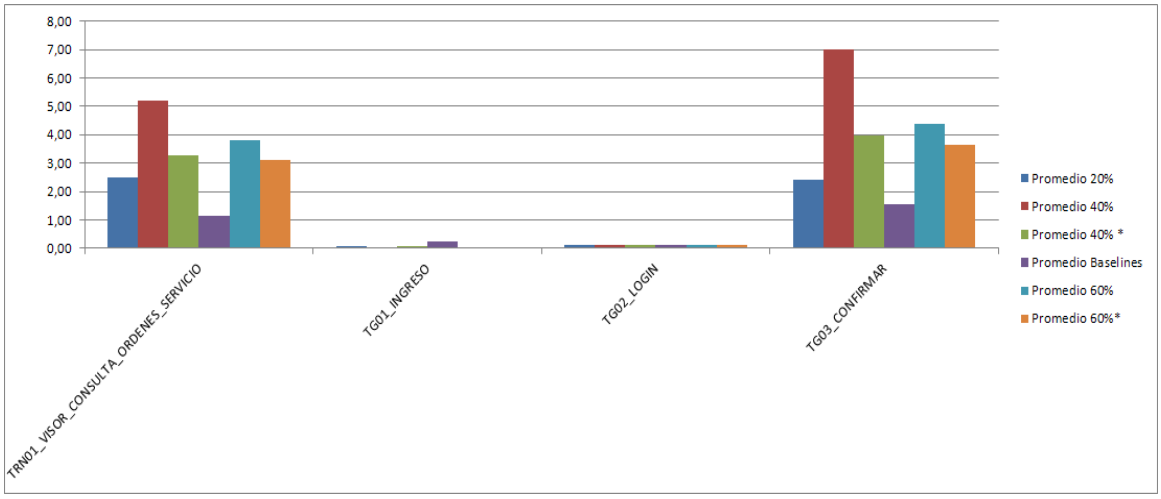

**Gráfica 99: Tiempos de respuesta para la TRN01 | prueba 60%**

Como se pueden ver, los pasos que determinan el tiempo total de la transacción se relacionan de manera directa con el visor. Ya sea la visualización de las órdenes o el paso de confirmación, ambos despliegan el visor de órdenes y son las que consumen mayor tiempo. Estos tiempos van desde un mínimo de menos de un segundo a un máximo de menos de 10 segundos (en una ráfaga momentánea), lo cual indica que la concurrencia o la combinación usuario / hospital influye sobre dicha transacción.

En cuanto a la comparación con los tiempos base, el promedio se puede ver en la gráfica anaranjada (abajo - Promedio 60%*) el cual sigue disminuye con respecto a la prueba del 60% y disminuye también incluso con respecto a la ejecución del 40%*.

El promedio total del paso de visualización de las órdenes provenientes de una confirmación supera por poco los tres segundos.

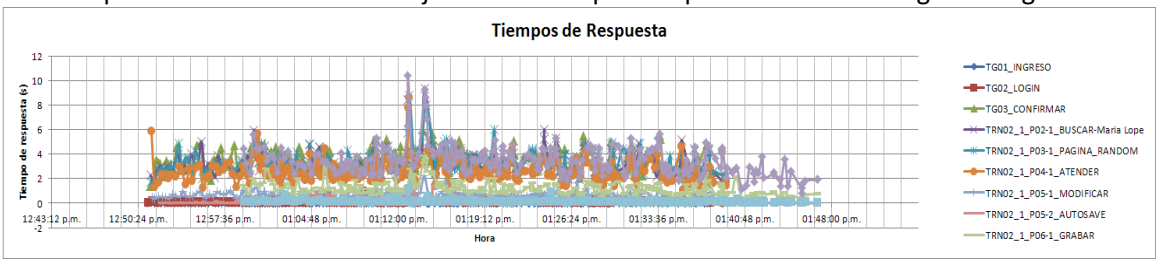
Los tiempos de respuesta promedio de los escenarios ejecutados se puede ver a continuación.




**Gráfica 100: Comparación tiempos de respuesta TRN01 | prueba 60%**

## TRN02_1_FICHAS_DINAMICAS_[INSERTAR_REGISTROS_ESPAÑOL]

Los tiempos obtenidos durante la ejecución de la prueba pueden verse en la gráfica siguiente

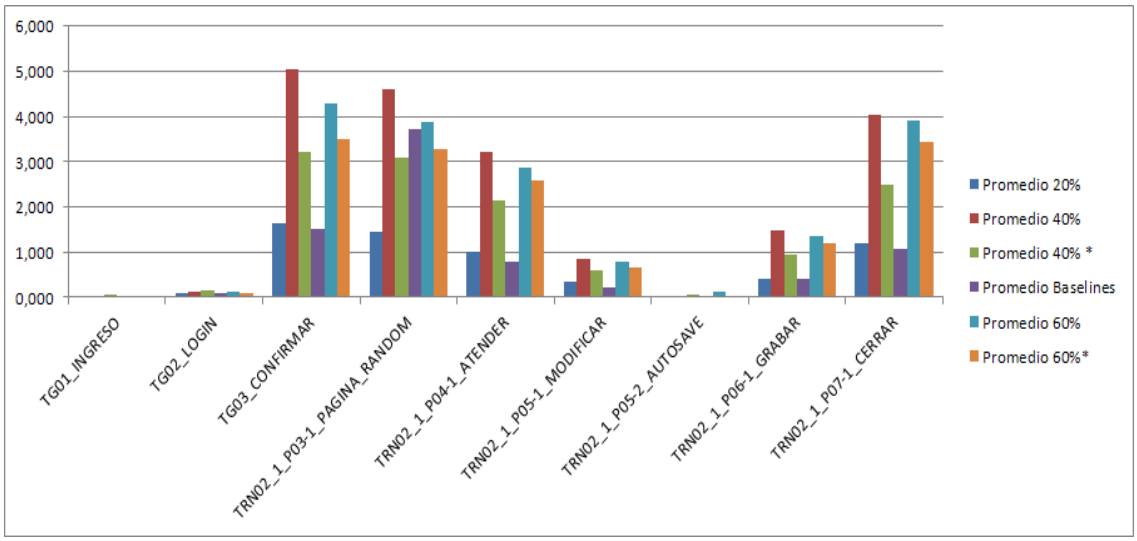



**Gráfica 101: Tiempos de respuesta para la TRN02_1 | prueba 60%**

En la gráfica se puede ver el mismo patrón de comportamiento en cuanto a los tiempos asociados al visor. Los tiempos de respuesta van desde 1 segundo y llegan a superar los 10 segundos. Los mejores tiempos se observan sobre el final de la prueba donde hay menos carga y concurrencia de usuarios y transacciones.

Los peores tiempos de respuestas se dan en una racha aislada y rondan los 10 segundos.

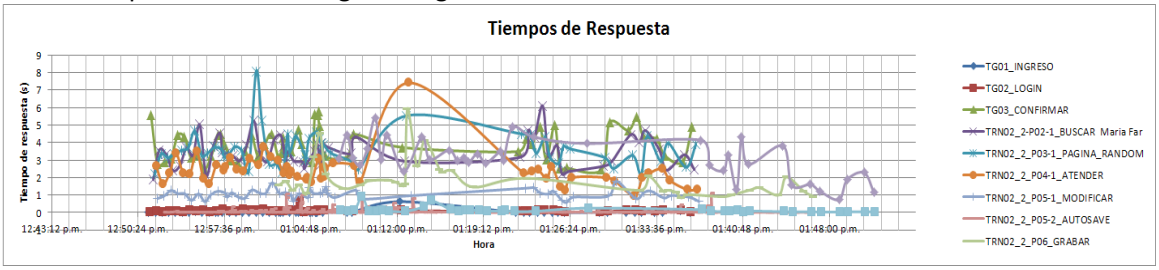
La comparación contra los tiempos base (en su promedio) pueden verse en la siguiente gráfica. Se una mejora de los tiempos con respecto a los obtenidos en la prueba del 60%. Pero en este caso no llegan a ser tan buenos como en la prueba del 40%*.




**Gráfica 102: Comparación tiempos de respuesta TRN02_1 | prueba 60%**

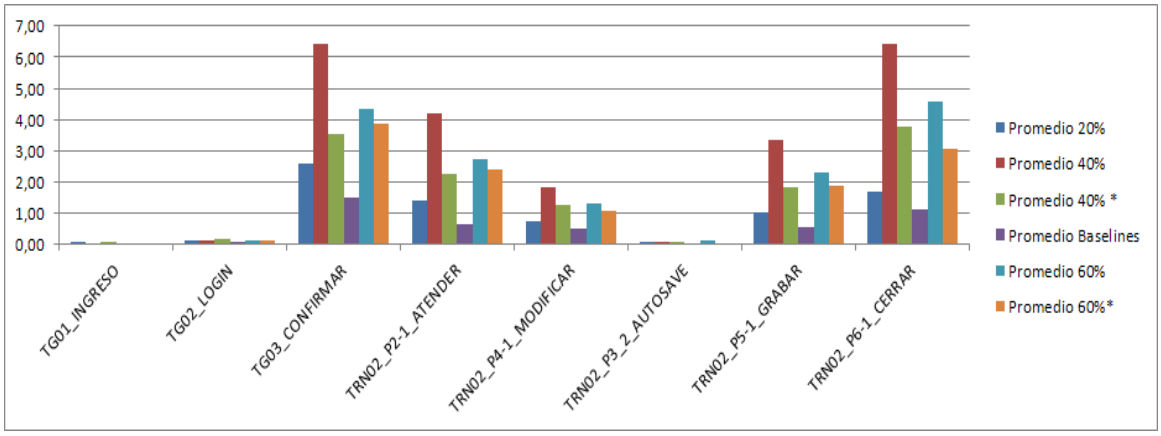
## TRN02_2_FICHAS_DINAMICAS_[INSERTAR_REGISTROS_PEREIRA]

Los tiempos que se dieron en la transacción de grabación de fichas dinámicas del tipo “Pereira Rossell” se puede ver en la siguiente gráfica:




**Gráfica 103: Tiempos de respuesta para la TRN02_2 | prueba 60%**

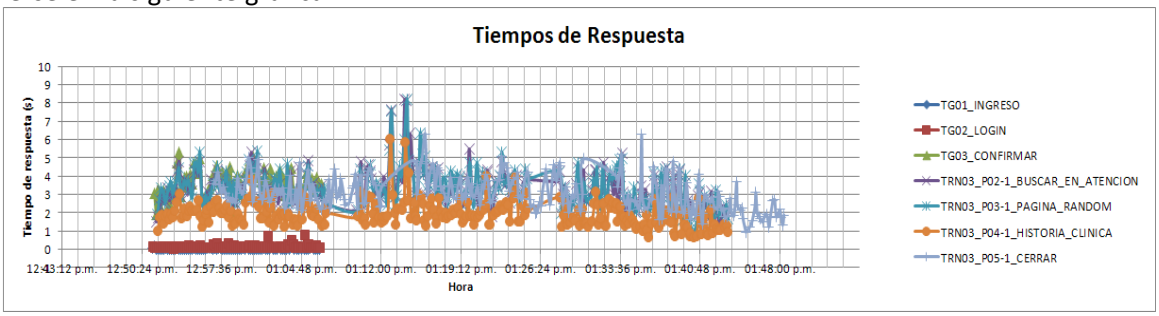
El comportamiento mencionado se repite en los tiempos de esta transacción. En comparación con los tiempos de los demás escenarios se puede notar la mejora respecto a la prueba del 60% y se podría decir que prácticamente se equipara con la prueba del 40%*.




**Gráfica 104: Comparación tiempos de respuesta TRN02_2 | prueba 60%**

## TRN03_HISTORIA_CLINICA_[VISUALIZACION]

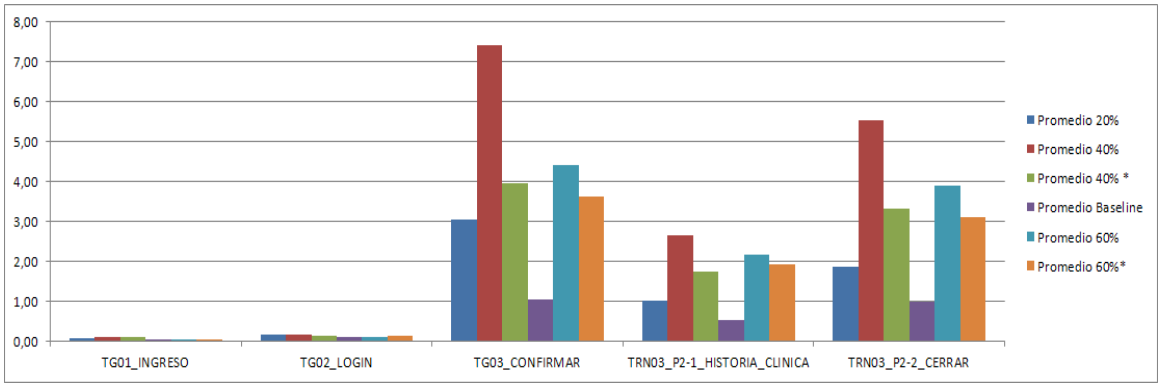
Los tiempos obtenidos para los diferentes pasos de la visualización de Historia Clínica pueden verse en la siguiente gráfica.




**Gráfica 105: Tiempos de respuesta para la TRN03 | prueba 60%**

Nuevamente se observa el mismo comportamiento y de manera coincidente con las otras transacciones. Esto confirma el comportamiento observado anteriormente.

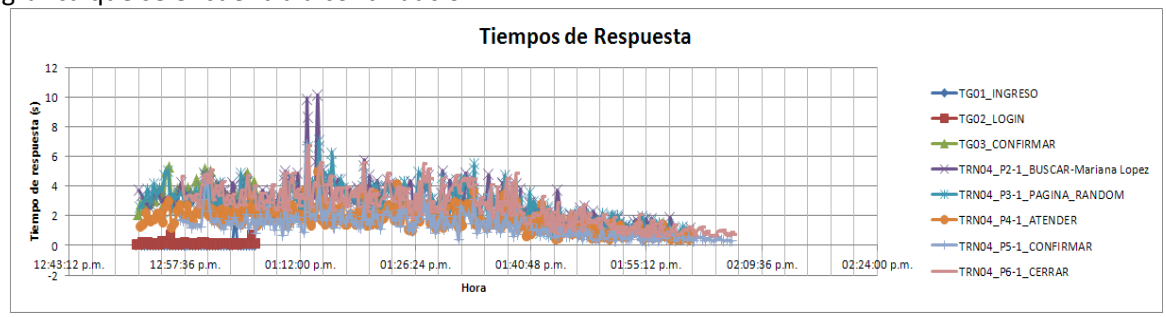
La comparación contra demás escenarios en cuanto al promedio arroja conclusiones similares a las transacciones anteriores.




**Gráfica 106: Comparación tiempos de respuesta TRN03 | prueba 60%**

### TRN04_ATENCION_[FINALIZAR]

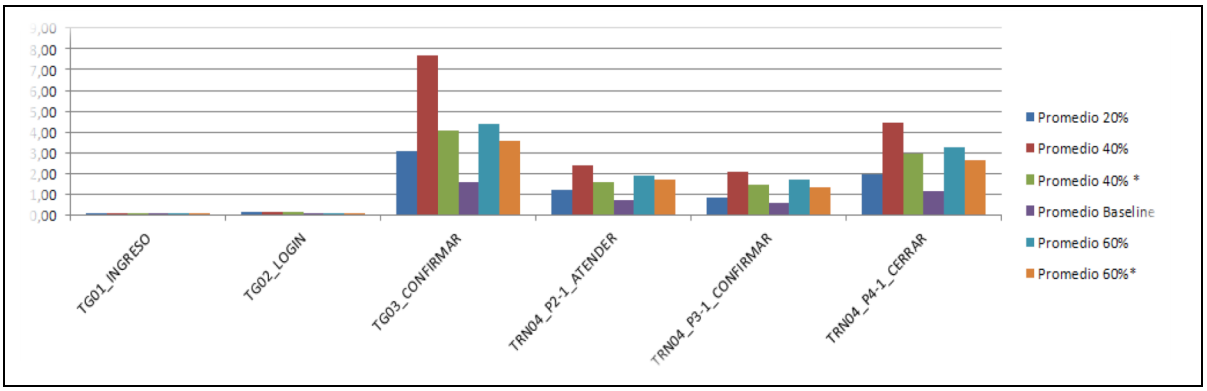
Los tiempos obtenidos para la transacción de finalización de atención pueden observarse en la gráfica que se encuentra a continuación.




**Gráfica 107: Tiempos de respuesta para la TRN04 | prueba 60%**

El patrón de la gráfica es similar al del resto de las transacciones, llegando los tiempos de respuesta a valores similares.

La comparación con los tiempos de los restantes escenarios ya ejecutados pueden verse en la siguiente gráfica.

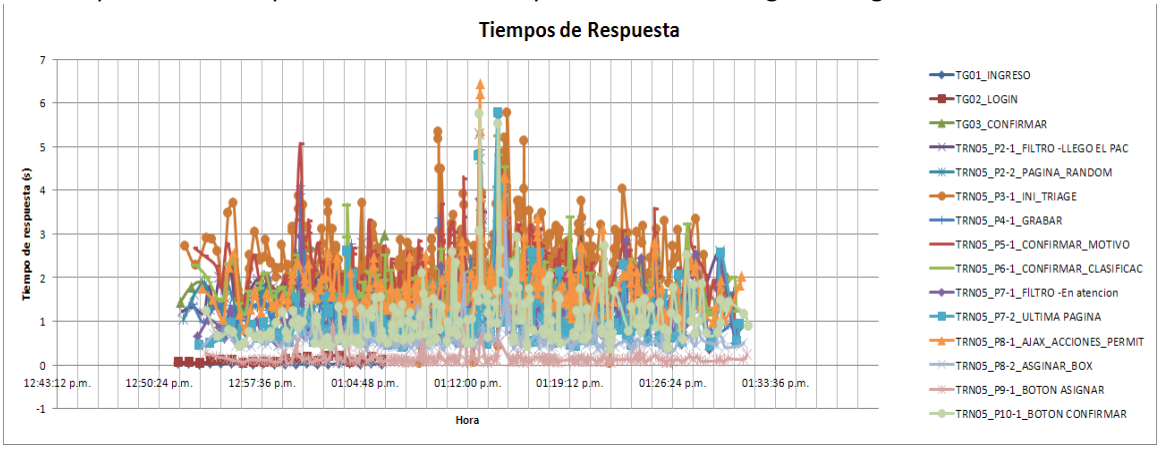



**Gráfica 108: Comparación tiempos de respuesta TRN04 | prueba 60%**

Nuevamente, los tiempos mejoran con respecto a la ejecución del 60 e incluso se encuentran en un nivel similar a la ejecución del 40%.

### TRN05_BOX_[ASIGNAR]

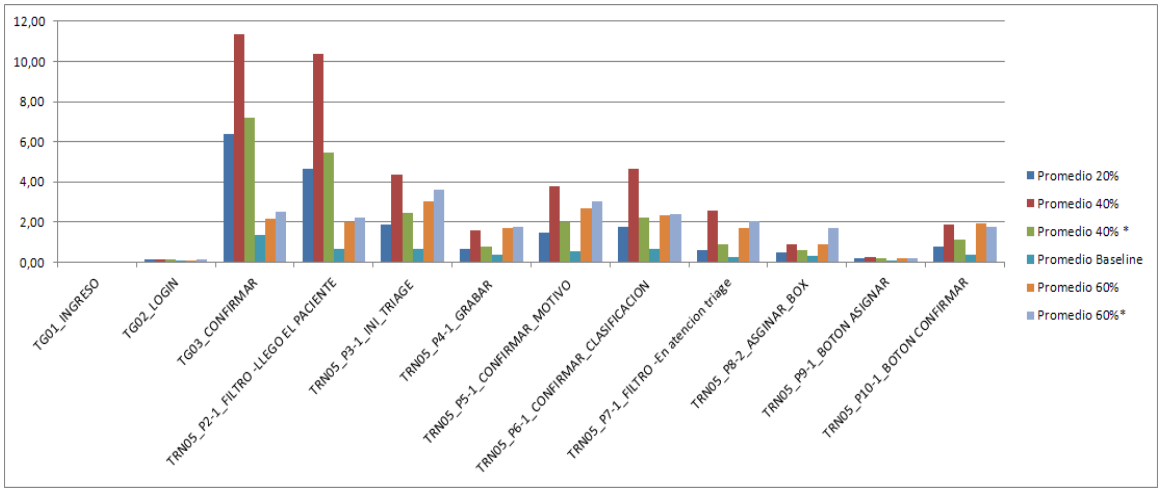
Los tiempos obtenidos para esta transacción pueden verse en la gráfica siguiente.




**Gráfica 109: Tiempos de respuesta para la TRN05 | prueba 60%**

Esta gráfica muestra tiempos similares a los obtenidos en las transacciones anteriores.

La comparación de los tiempos promedios de los distintos escenarios ejecutados para esta transacción puede verse en la siguiente gráfica.

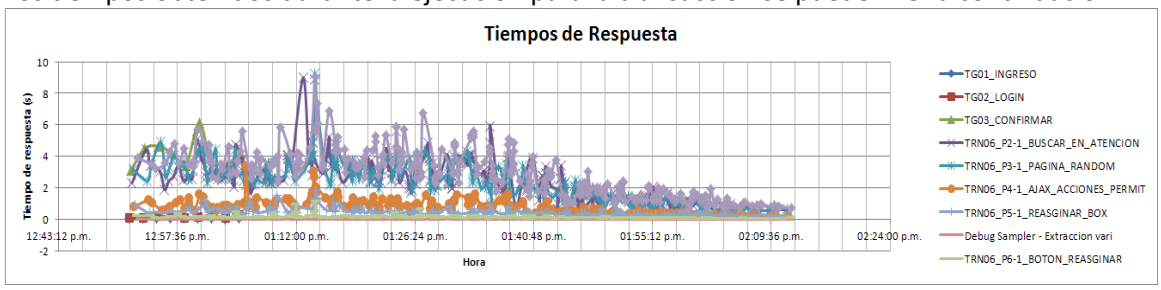



**Gráfica 110: Comparación tiempos de respuesta TRN05 | prueba 60%**

En esta gráfica se puede ver que la diferencia más grande obtenida con respecto a las ejecuciones anteriores, se da en las funcionalidades asociadas con el visor.

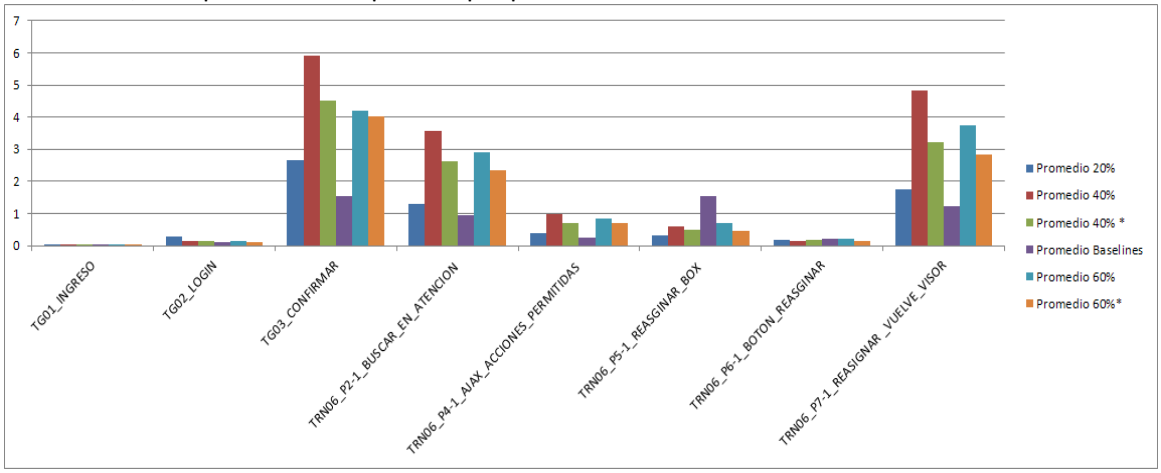
## TRN06_BOX_[REASIGNAR]

Los tiempos obtenidos durante la ejecución para la transacción se pueden ver a continuación.




**Gráfica 111: Tiempos de respuesta para la TRN06 | prueba 60%**

En esta transacción vuelven a verse los patrones que se repiten en todas las transacciones.
Comparando los tiempos promedios obtenidos en el escenario del 60%* contra los otros escenarios, se repite el mismo patrón que para las restantes transacciones.

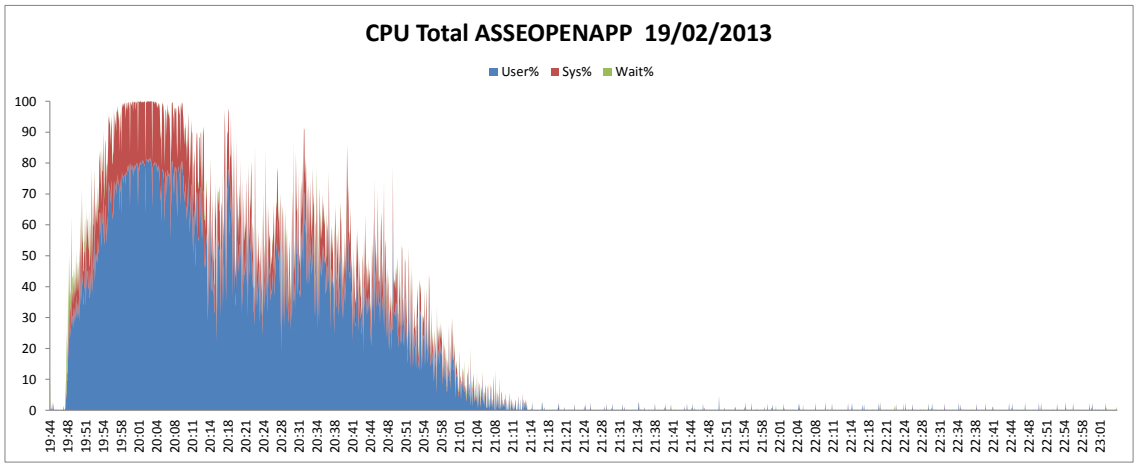



**Gráfica 112: Comparación tiempos de respuesta TRN06 | prueba 60%**

#### 5.5.2.5. Uso de la infraestructura

Durante la ejecución de las pruebas, el uso de la infraestructura no ha sido un cuello de botella salvo en las primeras ejecuciones del 40% y el 60%. En ambos casos, la infraestructura demostró ser un cuello de botella en lo que respecta al uso de los recursos en el servidor de base de datos. El servidor de aplicaciones demostró no ser un cuello de botella en la ejecución de las pruebas.

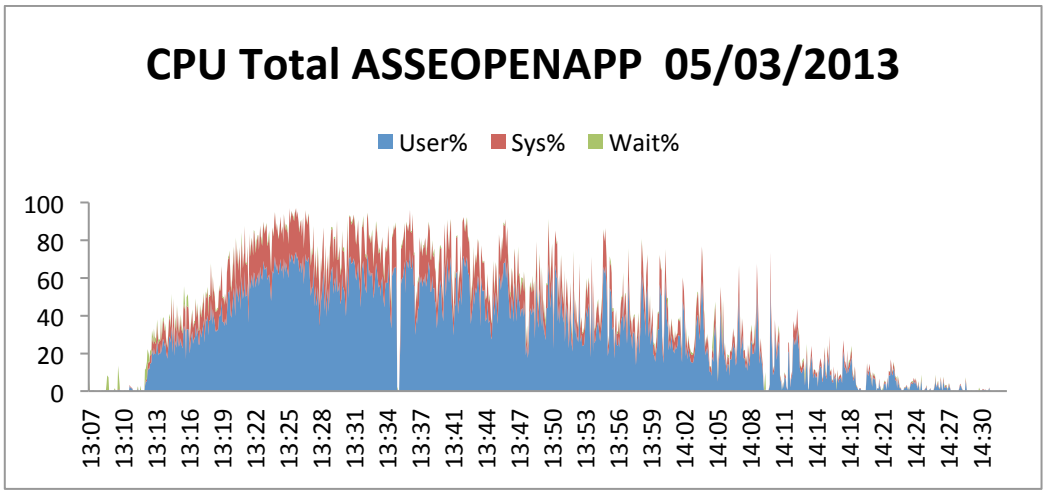
A modo de ejemplo, en la primera prueba del 40%, se obtuvo la siguiente gráfica de uso del CPU:




**Gráfica 113: Uso de CPU en DBMS previo agregar recursos - 40%**

Aquí se puede ver que en los primeros minutos de las pruebas, el uso de CPU llegó al 100%, teniendo un tiempo de usuario de más del 70%.

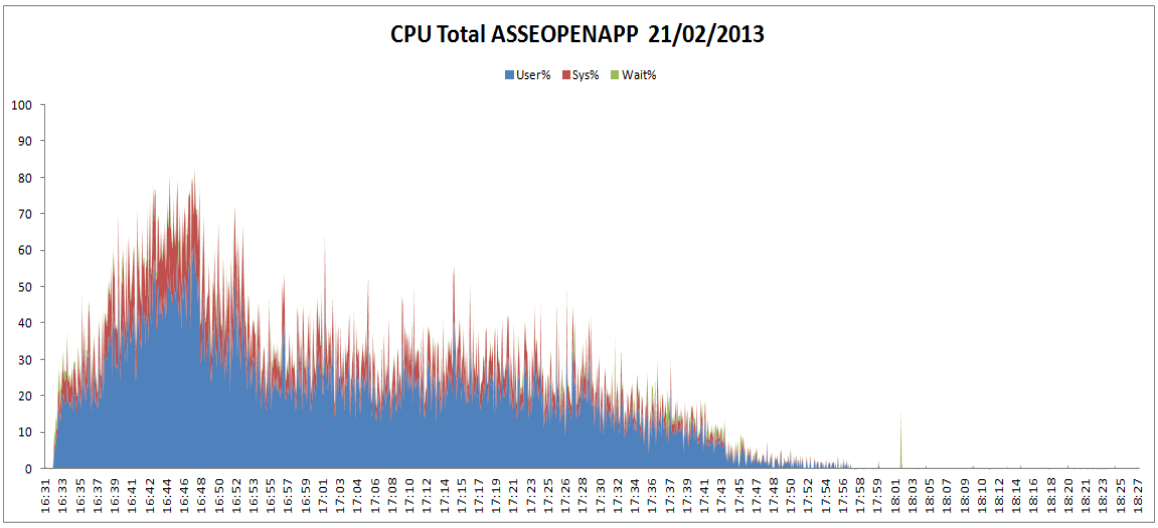
Algo similar sucedió a la hora de ejecutar la primera prueba del 60%, teniendo un uso de CPU que se documenta en la siguiente gráfica:




**Gráfica 114: Uso de CPU en DBMS previo *tunning* - 60%**

Allí se puede ver como tenemos un uso de recursos superior al 80% en lo referente a la CPU. Asimismo, esta monitorización permitió visualizar una pausa en el uso de los recursos que coincidió con procesos internos de la base de datos.

En el primer caso, se agregó mayor cantidad de recursos al servidor de base de datos y en el segundo se realizó el ajuste de los parámetros de la base de datos que pueden verse enumerados al inicio de la descripción de las pruebas del 60%. En el primero de los casos, agregar mayor cantidad de CPU redundó en una mejora inmediata de los tiempos y en una utilización directa de los recursos por parte del servidor de base de datos.




**Gráfica 115: Uso de CPU en DBMS post agregar recursos - 40%**

A partir de los resultados de las pruebas realizadas, se generó la siguiente tabla que enumera los recursos utilizados por la aplicación y la base de datos en los diferentes escenarios. Esta tabla puede usarse para estimar los recursos necesarios según la carga que se le impondrá al sistema.

| | Memoria Utilizada (MB) - BD | Memoria Utilizada (MB) - App |
|---|---|---|
| Escenario 20% | 1880 | 3800 |
| Escenario 40% | 3900 | 3500 |
| Escenario 60% | 1000 | 2500 |
| Escenario 60%* | 2800 | 2600 |

#### 5.5.2.6. Pruebas con cargas mayores

Se ejecutaron pruebas tendientes a alcanzar el 80% de la carga. Al momento de finalizar el proyecto, estas cargas no arrojaron tiempos aceptables según los criterios definidos.

El *testware* generado fue entregado a las organizaciones que participaron del proyecto y se brindó una capacitación para poder continuar con la ejecución de los escenarios. Meses más adelante nos notificaron que luego de varios ajustes la carga había sido soportada. Posteriormente el sistema fue puesto en producción.

### *5.5.3. Conclusiones*

Se obtuvo una prueba exitosa para el 60% de la carga objetivo. Al momento de finalizar el proyecto no fue posible alcanzar el 100% de la carga objetivo. Los distintos escenarios de infraestructura planteados no soportaron la carga. En el escenario definido junto a ASSE (un total de 46 hospitales) esta carga podría llegar a darse.

Hay que destacar que se partió de un escenario de ejecución de tiempos bases inaceptables y gracias a las mejoras impuestas al software se ejecutó un escenario del 60% satisfactorio y con una infraestructura que permite un margen de crecimiento.

Las pruebas efectuadas han sido satisfactorias en el aspecto de que permitieron mejorar sustancialmente la performance de la aplicación, optimizando para su lógica.

Se planteó la colaboración para alcanzar las cargas del 100% y poder ejecutar las mismas en la infraestructura de prueba y producción planteada por ASSE. Esto se logró y de esta manera ASSE tuvo la tranquilidad de que la aplicación cumplirá con los requerimientos no funcionales en su escenario de infraestructura, permitiendo ello seguir con el plan de implantación definido.

# 6. Conclusiones

En este capitulo se resume brevemente los resultados obtenidos y las contribuciones. A su vez, se detallan los posibles trabajos futuros a realizar y finalmente las conclusiones.

## 6.1. Resultados y contribuciones

Los principales resultados y contribuciones obtenidos son: el diseño, puesta en práctica y mejora de una metodología para realizar pruebas de rendimiento. La correcta aplicación de la misma brinda buenos resultados, entre ellos se destaca la predicción del comportamiento de SUTs ante situaciones de carga que se deseen simular sin necesidad que los usuarios reales participen.

## 6.2. Trabajos a futuro

Como trabajo a futuro se plantea mejorar el modelo de estimación de carga esperada. Es decir la definición de escenarios. Los mismos hoy se basan en simples reglas de tres a partir de datos recolectados, muchas veces el único sustento es la palabra de un funcional y el conocimiento del negocio que alguien pueda tener. En otros casos se cuenta con acceso a registros del sistema (*logs*). El ideal es reproducir el uso del sistema y poder escalar de forma consistente la carga.

Se propondrán proyectos de grado para considerar la mejora de estos temas. Una primer versión propuesta y prototipada (Barbatto & Díaz, 2014) que se trabajó asume el acceso a registros del sistema en formato W3C obtenidos del servidor de aplicación por un período que capture satisfactoriamente el uso real de la aplicación y que permite estimar un modelo de Markov de largo variable, generando simulaciones que son estadísticamente similares al comportamiento de usuarios reales. La misma se basa en la generación de escenarios realistas del uso del sistema. Esta técnica permite, por ejemplo, probar diferentes escenarios de expansión de cantidad de usuarios, o ensayar plataformas de hardware alternativas para un sistema. La principal ventaja que obtenemos de la aplicación de esta metodología y herramientas es la de generar escenarios realistas que están basados en el uso real de la aplicación. Otra alternativa y también deseable es estimarla a partir de una base de conocimiento de sistemas similares o construir un marco de trabajo para cada organización.

## 6.3. Conclusiones

La necesidad y el alcance de realizar pruebas a un sistema depende de la criticidad del mismo. Las buenas pruebas brindan información sobre la calidad del sistema y permite nivelar las expectativas. La información sobre la calidad es útil para mejorarla. Requiere el análisis de distintos expertos.

Existen distintos tipos de pruebas de rendimiento. Su aplicación dependerá de la arquitectura y los objetivos que se persiguen.

La ingeniería de performance permite mejorar el rendimiento de los sistemas. Para mejorar el sistema rápidamente generalmente se necesita un equipo multidisciplinario que tenga conocimiento de cada una de las tecnologías del SUT (ej. base de datos, servidores de aplicación, entre otros).

La aplicación de pruebas de rendimiento junto con ingeniería de performance permite mejorar el rendimiento de los sistemas previo a que el sistema esté en producción o analizar las causas y reproducir los problemas en un ambiente controlado.

En esta tesis se introduce una nueva metodología para pruebas de rendimiento. Su eficiencia fue puesta en práctica en varios casos reales de sistemas críticos de salud, banco y *contact center*, los resultados se pueden ver en los casos citados. Se obtuvo información sobre el rendimiento del sistema y aplicando ingeniería de performance se logró mejorar varios de los problemas detectados en tiempo y forma.

La metodología presentada en este documento fue probada y se obtienen buenos resultados, que permiten predecir el comportamiento del SUT ante situaciones similares de carga.

# 7. Anexo I - Requisitos

A continuación, se mencionan los requisitos que son necesarios en los plazos previstos en el plan del proyecto para poder realizar una prueba de estás características:

- Al inicio del proyecto se deberá contar con una versión congelada del sistema, funcionalmente probada y estable, instalado y configurado en equipos con suficientes recursos tales que permiten ejecutar de manera correcta la prueba y está disponibles a los "testers".
- Acceso a los datos o las consultas en SQL a ser usadas para generar los datos de los parámetros de aquellos scripts que lo requieran.
- Definición de las métricas de performance y valores objetivos para el SUT.
- Generación del volumen y calidad de datos necesarios para las pruebas.
- Participación de analistas funcionales y desarrolladores en la revisión y validación de las transacciones del sistema a probar, en la validación de la instalación realizada y en la validación de los escenarios que se propongan para las pruebas.
- Participación de analistas funcionales, desarrolladores y especialistas técnicos durante la etapa de ejecución de pruebas.
- Contar con herramientas de monitoreo de la infraestructura sobre la que se despliega el sistema y con el soporte de especialistas en el hardware y su sistema operativo para resolver aspectos relacionados con su configuración y "tuning".
- Contar personal que sea capaz de configurar o agregar niveles de *logs* necesarios.

# 8. Anexo II - Tareas y roles

## 8.1. Equipo de pruebas

### *8.1.1. Líder de Proyecto*

- Definir los criterios de aceptación para el proyecto y los criterios de finalización del mismo, junto con el responsable del proyecto.
- Definir los objetivos, requerimientos y alcance del proyecto de la prueba de desempeño, junto con el responsable del proyecto y el equipo de pruebas.
- Realizar y presentar la propuesta y el plan del proyecto de pruebas.
- Identificar junto con el responsable del proyecto y el equipo de pruebas las transacciones y escenarios más sensibles a las pruebas de desempeño y las que tengan mayor probabilidad de no cumplir con el desempeño requerido.
- Definir junto con el responsable del proyecto y el equipo de pruebas, las métricas de desempeño y valores objetivos para las transacciones bajo los escenarios identificados.
- Darle visibilidad al proyecto.
- Supervisar la verificación de la realización de pruebas funcionales y la corrección de los defectos encontrados.
- Acordar con el responsable del proyecto el procedimiento a seguir en caso de aparición de defectos que impidan continuar con las pruebas.
- Validar la factibilidad técnica del uso de la herramienta de generación de carga.
- Supervisar la automatización de las pruebas
- Validar el armado del ambiente de prueba y su preservación.
- Supervisar la ejecución de las pruebas.
- Interactuar con el responsable del proyecto y el equipo de pruebas en todas las etapas técnicas de la ejecución del proyecto.
- Asegurar la calidad de todos los productos del proyecto de prueba.
- Analizar junto al equipo de pruebas los resultados de las pruebas.
- Reportar los resultados (ej. Mediante informes).

### *8.1.2. Tester senior*

- Participar en la identificación de las transacciones y escenarios más sensibles a las pruebas de desempeño y las que tengan mayor probabilidad de no cumplir con el desempeño requerido.
- Verificar la factibilidad técnica del uso de la herramienta de generación de carga.
- Elaborar los reportes de prueba.
- Verificar la automatización y la ejecución de las pruebas.

### *8.1.3. Automatizador/Tester*

- Captura y parametrización de las transacciones en la herramienta a utilizar.
- Ejecutar las transacciones automatizadas según los escenarios definidos en la etapa de diseño.
- Recolectar los indicadores necesarios que permitan la evaluación del desempeño del sistema.
- Analizar los resultados de las pruebas realizadas.
- Verificar la corrección de aquellos factores que afecten la correcta realización de la prueba.

## 8.2. Equipo de SUT

### *8.2.1.Responsable del proyecto*

- Definir los objetivos, requisitos y alcance del proyecto de la prueba de desempeño, junto con el líder del proyecto.
- Definir los criterios de aceptación para el proyecto y los criterios de finalización del mismo, junto con el líder del proyecto.
- Acordar con el líder del proyecto el procedimiento a seguir en caso de aparición de defectos que impidan continuar con las pruebas.
- Identificar junto con el equipo de pruebas las transacciones y escenarios más sensibles a las pruebas de desempeño y las que tengan mayor probabilidad de no cumplir con el desempeño requerido.
- Definir junto con el líder del proyecto la forma en que se reportan los incidentes encontrados y quienes serán los responsables por parte del cliente de solucionar los defectos.
- Brindar acceso a la documentación requerida (documentos de requerimientos, manuales, documentación técnica, usuarios del sistema, resultados de la prueba de concepto de la herramienta, resultados de las pruebas funcionales, entre otros) para la realización de la prueba.
- Interactuar con el equipo de pruebas en todas las etapas técnicas de la ejecución del proyecto.
- Identificar los responsables de validar y corregir los incidentes encontrados en las pruebas y garantizar su correcta y oportuna solución.
- Validar todas las definiciones realizadas por el equipo de pruebas. Esto incluye los escenarios, transacciones e infraestructura.
- Validar los informes entregados.

### *8.2.2.Responsable técnico de la aplicación*

- Brindar los datos necesarios para la ejecución de las pruebas.
- Identificar, junto al equipo de pruebas, las transacciones y escenarios más sensibles a las pruebas de desempeño y las que tengan mayor probabilidad de no cumplir con el rendimiento requerido.
- Realizar el seguimiento y validación de los incidentes detectados por el equipo de pruebas.
- Responsabilizarse por la corrección de errores detectados durante la realización de la prueba.
- Estar disponible para contestar dudas respecto a aspectos técnicos del sistema.

### *8.2.3.Responsable de la infraestructura del sistema*

- Definir junto con el líder del proyecto, las métricas de desempeño y valores objetivos para las transacciones bajo los escenarios identificados.
- Administrar el entorno de prueba.
- Asistir en la generación el ambiente para las pruebas, instalación y configuración de la infraestructura necesaria (hardware y software)
- Estar disponible para contestar dudas respecto a aspectos técnicos del sistema e infraestructura.
- Validar la definición de la infraestructura realizada por el equipo de pruebas.

### *8.2.4.Responsable funcional de la aplicación*

- Identificar, junto al equipo de pruebas, las transacciones y escenarios más sensibles a las pruebas de desempeño y las que tengan mayor probabilidad de no cumplir con el desempeño requerido.
- Brindar los datos necesarios para la automatización de las pruebas.
- Estar disponible para contestar dudas respecto a aspectos funcionales del sistema.
- Validar las definiciones de transacciones realizadas por el equipo de pruebas.

# 9. Anexo III – Guiones fusión

*TRN01 - Retiro en cuentas pasivas*

| Realizar acción | Resultado | Think time |
|---|---|---|
| Login (Cajero) | | Bajo |
| Seleccionar el menú "Inicio -> Menú de cajas -> Operaciones de Caja" | Va a la pantalla de Operaciones de Caja | Bajo |
| Ingresar Módulo = 50 y Transacción = 423 (para retiro en soles) y clic en Confirmar | Va a la pantalla de Efectivo en Agencias y Ventana | Bajo |
| Ingresar Importe y presionar "Confirmar" | Se va a la pantalla de ingreso de Cuentas de Ahorros | Bajo |
| Ingresar número de cuenta, moneda = 0 (soles), sub-cuenta con disponible y presionar "Continuar" | Va a la pantalla de Consulta de Firmas | Medio |
| Presionar el botón de "Cerrar" | Va a la pantalla de Trabajar con Gastos | Bajo |
| Presionar botón de "Cerrar" | Va a la pantalla de confirmación de la transacción | Bajo |
| Presionar el botón "Confirmar" | Se va a la pantalla de Intervención de Documentos (pero no se debe imprimir) | Bajo |

*TRN02 - Depósito en cuentas pasivas*

| Realizar acción | Resultado | Think time |
|---|---|---|
| Login (Cajero) | | Bajo |
| Seleccionar el menú "Inicio -> Menú de cajas -> Operaciones de Cajas" | Va a la pantalla de Operaciones de Caja | Bajo |
| Seleccionar el submenú "501 - Depósito Efectivo Soles S/." | Va a la pantalla de "Depósito Efectivo Soles S/." | Bajo |
| Ingresar la Cuenta, Suboperación a consultar e Importe) y presionar en "Validar" | Carga el nombre en el value asociado a "_NOM_PREF_1_0001" | Medio |
| Presionar el botón "Confirmar" | Va a la pantalla de "Trabajar con Gastos" | Bajo |
| Presionar el botón "Cerrar" | Va a la pantalla de confirmación, mostrando en rojo la palabra "CONFIRMACION" | Bajo |
| Presionar el botón "Confirmar" | Va a la pantalla de "Intervención de Documentos" | Bajo |

*TRN03 – Alta de plazo fijo*

Ver sección “Transacciones seleccionadas” del caso en cuestión (“Fusión de dos bancos”).

*TRN04 – Confirmación de alta de plazo fijo*

| Realizar acción | Resultado | Think time |
|---|---|---|
| Login (Cajero) | | Bajo |
| Seleccionar el menú "Inicio -> Menú de Caja -> Confirmación de transacciones" | Va a la pantalla de "Confirmación de Transacciones" | Bajo |
| Clic sobre el primer elemento y presionar "Seleccionar" | Se selecciona el elemento | Medio |
| Clic en "Confirmar" | Va a la pantalla de "Intervención de documentos" | Bajo |
| Clic en "Confirmar" | Vuelve a la pantalla: "Confirmación de Transacciones" | Bajo |

*TRN05 – Cancelación de plazo fijo*

No se incluyó en las pruebas

*TRN06 – Solicitud de crédito*

| Realizar acción | Resultado | Think time |
|---|---|---|
| Login (Asesor) | | Bajo |
| Seleccionar el menú "Inicio -> Menú de Flujo de Tareas -> Inbox Workflow" | Se muestra la pantalla "Bandeja de Entrada de Tareas" | Bajo |
| Clic en iniciar proceso (abajo del todo) | Va a la pantalla de "Iniciar Instancia de Proceso" | Bajo |
| Clic en "solicitud de crédito" y en "iniciar" | Va a la pantalla de "Solicitud de Crédito." | Bajo |
| Ingreso Nro. de cuenta que sea de un cliente del segmento independiente y presiono "seleccionar" | Se seleccionan los datos del cliente | Bajo |
| Presiono "Confirmar" | Va a la pantalla de "Solicitud de Crédito." con el cliente seleccionado | Bajo |
| Hago clic en selección de producto -> Producto | | Bajo |
| Selecciono de Datos de Crédito -> "Módulo" = "104-NEGOCIOS" | Aparece el Tipo de operación | Bajo |
| Elijo el tipo de operación = 2 (cuotas) y hago clic en seleccionar | | Bajo |
| Ingreso cantidad de cuotas, periodo de cuotas, Monto, fecha valor = fecha del sistema, fecha del primer pago = al menos 30 días posterior a fecha (capaz es importante cambiar la fecha previo a cada prueba), Destino del Crédito "Capital de Trabajo", (carga Tasa) y hago clic en seguros | Va a la página "Seguros de la Operación" | Medio |
| Clic en "De Inmueble Contra Todo | Cambia de pantalla | Medio |

| | | |
|---|---|---|
| Riesgo/Casa Hab" y clic en "Eliminar" | | |
| Presiono continuar | Retorna | Bajo |
| Presionar comisiones | Va a la pantalla "Comisiones de la Operación" | Bajo |
| Hago clic en los checks y clic en eliminar | Quita todos | Medio |
| Clic en confirmar | Retorna | Bajo |
| Clic en "plan de pagos" | Va a la pantalla "Plan de Pagos Amortizable" | Bajo |
| Clic en volver | Retorna | Bajo |
| Clic en aceptar | Retorna | Bajo |
| Ingresar cuenta (de garantes 465492) y hago clic en añadir | Agrega al garante | Medio |
| Clic en "Datos Adic." | Va a la pantalla "Solicitud de Créditos" con "Sub-Tipo de Crédito" | Bajo |
| Seleccionar el Sub-Tipo = "CLASICO" y clic en confirmar | Retorna | Bajo |
| Clic en grabar | Retorna | Bajo |
| Clic en Aceptar (queda en la etapa Evaluación / Propuesta) | Va a la pantalla "Bandeja de Entrada de Tareas" (Si va a la pantalla "Validación de Políticas" avisar pues hay que cambiar la configuración) | Bajo |

*TRN07 – Evaluación de créditos*

| **Realizar acción** | **Resultado** | **Think time** |
|---|---|---|
| **Asesor** | | Bajo |
| Seleccionar el menú "Inicio -> Menú de Flujo de Tareas -> Inbox Workflow" | Se muestra la pantalla "Bandeja de Entrada de Tareas" | Bajo |
| Ejecutar la instancia (selecciono y clic en "ejecutar") | Va a la pantalla: "Evaluación Crediticia - Información Socioeconómica" | Bajo |
| Clic en "ctas/por/cobrar" (abajo) | Va a la pantalla "Cuentas por Cobrar" | Bajo |
| Fecha = CES_FECHA; Cliente=CES_CLIENTE;mda. Nac = 500; clic en grabar | Sale el total | Medio |
| Clic en volver | Vuelve a "ESTADO DE GANANCIAS Y PERDIDAS" Y CARGA VALORES | Bajo |
| Clic en "inventarios"(abajo) | Va a la pantalla | Bajo |

| | | |
|---|---|---|
| | "Inventarios | |
| cantidad = 1000;articulos=ces_sacos_de_azucar, unidad de medida=ces_sacos, costo unidad moneda nacional = 50, clic en grabar | Sale el total en moneda nacional | Medio |
| clic en grabar (si, 2 veces :)) | Sale el total | Bajo |
| Clic en volver | Vuelve a "ESTADO DE GANANCIAS Y PERDIDAS" Y CARGA VALORES | Bajo |
| Clic en "activos"(abajo) | Va a la pantalla "Activos Fijos" | Bajo |
| Cantidad =1 Descripción "balanza" Estado de Conserv. "bueno" Antigüedad (Años) "5" Valor Merc Mda Nac. "2000"<br>Cantidad =1 Descripción "triciclo" Estado de Conserv. "bueno" Antigüedad (Años) "5" Valor Merc Mda Nac. "1500"<br><br>clic en grabar ABAJO (1 sola vez) | Sale el total DE ABAJO | Medio |
| Cantidad "1" Descripción "local comercial" Propiedad Legal "minuta..." Estado de Conserv. "buena" Antigüedad (Años) = "10" Valor Merc. Mda Nac. "20.000" Valor<br><br>Clic en grabar ARRIBA | Sale el total DE ARRIBA | Medio |
| Clic en volver | Vuelve a "ESTADO DE GANANCIAS Y PERDIDAS" Y CARGA VALORES | Bajo |
| Clic en cuentas por pagar | Va a la pantalla: "Cuentas por Pagar" | Bajo |
| Ingresamos ces_texto_generico en los campos menos monto que ponemos 600 y clic en grabar | Actualiza datos | Medio |
| Clic en volver | Vuelve a "ESTADO DE GANANCIAS Y PERDIDAS" Y CARGA VALORES | Bajo |
| Clic en deudas financieras | Va a la pagina: "Deudas Financieras" | Bajo |
| ingresamos cuota mensual "300" y saldo capital = "2500" clic en grabar | Actualiza datos | Medio |
| Clic en volver | Vuelve a "ESTADO DE GANANCIAS Y PERDIDAS" Y CARGA VALORES | Bajo |
| Clic en "Gtos. Familiares" | Va a la pantalla: "Gastos Familiares" | Bajo |
| texto = ces_texto_generico; en mda. Nacional = "2500" y clic en grabar | Actualiza datos | Medio |
| Clic en volver | Vuelve a "ESTADO DE GANANCIAS Y PERDIDAS" Y CARGA VALORES | Bajo |

| | | |
|---|---|---|
| Clic en "gtas. operativos" | Va a la pantalla de "Gastos Operativos" | Bajo |
| concepto = ces_texto_generico; nro=1, costo unitario = 800, y clic en grabar | Actualiza datos | Medio |
| Clic en volver | Vuelve a "ESTADO DE GANANCIAS Y PERDIDAS" Y CARGA VALORES | Bajo |
| Clic en "margen utilidad bruta" | Va a la pantalla de "Gastos Operativos" | Bajo |
| artículos = ces_texto_generico; unidad de medida = ces_texto_generico; pre-compra = 50; pre-venta = 60; %ventas = 100 y clic en grabar | Actualiza datos | Medio |
| Clic en volver | Vuelve a "ESTADO DE GANANCIAS Y PERDIDAS" Y CARGA VALORES | Bajo |
| Clic en "Otras Deudas Financieras" | Va a la pantalla de "Otras Deudas Financieras" | Bajo |
| monto = 1000.000, saldo = 800.00, cuota = 100.00, Nro cuotas/total = "2/12" y clic en grabar | Actualiza datos | Medio |
| Clic en volver | Vuelve a "ESTADO DE GANANCIAS Y PERDIDAS" Y CARGA VALORES | Bajo |
| Clic en disponible | Va a la pantalla de "Disponible | Bajo |
| total moneda nacional = 1500 y clic en grabar | Actualiza datos | Bajo |
| Clic en volver | Vuelve a "ESTADO DE GANANCIAS Y PERDIDAS" Y CARGA VALORES | Bajo |
| Clic en "Ventas" | Va a la pantalla de "vENTAS" | Bajo |
| frecuencia mensual = 1, ventas moneda nacional = 40000 textos genéricos, clic grabarX2 | Actualiza datos | Medio |
| Clic grabarX2 | Actualiza datos | Bajo |
| Clic en volver | Vuelve a "ESTADO DE GANANCIAS Y PERDIDAS" Y CARGA VALORES | Bajo |
| Clic en "Otros ingresos" | Va a la pantalla de "Otros ingresos" | Bajo |
| alquiler = 300, clic en grabar | Actualiza datos | Bajo |
| Clic en volver | Vuelve a "ESTADO DE GANANCIAS Y PERDIDAS" Y CARGA VALORES | Bajo |
| Clic en "Compras" | Va a la pantalla de "Compras" | Bajo |
| frecuencia = 1, compras = 30000 y grabar | Actualiza datos | Bajo |

| | | |
|---|---|---|
| Clic en volver | Vuelve a "ESTADO DE GANANCIAS Y PERDIDAS" Y CARGA VALORES | Bajo |
| Clic en grabar | "graba" | Bajo |
| Clic en "Ratios" | Va a la pantalla de "Ratios Microempresarios" | Bajo |
| Clic en volver | Vuelve a "ESTADO DE GANANCIAS Y PERDIDAS" Y CARGA VALORES | Bajo |
| Clic en grabar | se habilita propuesta | Bajo |
| Clic en propuesta | Va a la pagina: "Propuesta" | Bajo |
| textos genéricos y grabar | "graba" | Bajo |
| Aceptar | Se muestra la pantalla "Bandeja de Entrada de Tareas" | Bajo |
| Login:TCHER002 | | Bajo |
| inicio -> menú de consultas -> Consultas de excepciones | Va a la pantalla de consulta de excepciones | Bajo |
| En combo regional seleccionar 8 - tacna<br>En combo agencia seleccionar -Todas-<br>Presionar filtrar | Aparecen los pedidos | Bajo |
| Seleccionar la primera y presionar tomar | Va a la pantalla de comentario a excepción | Bajo |
| Ingresar Ces_comentario y presionar autorizar | Aparece la pantalla de login | Bajo |
| Ingresar clave y dar aceptar | Va a bandeja de entradas | Bajo |

*TRN08 – Aprobación de créditos*

No se incluyó en las pruebas

*TRN09 – Desembolso de créditos*

No se incluyó en las pruebas

*TRN10 – Amortización (cancelación) de créditos*

| **Realizar acción** | **Resultado** | **Think time** |
|---|---|---|
| **Login (Cajero)** | | Bajo |
| Seleccionar el menú "Inicio -> Menú de Caja -> Operaciones de Caja" | Se expande el menú para realizar consultas | Bajo |
| Ingresa módulo = 50, transacción = 140 y presionar confirmar | Va a la pantalla Selección de Operaciones | Bajo |
| Ingresar el nro. de la cuenta 235439<br>y presionar filtrar | Despliega la información para dicha cuenta | Bajo |
| Presionar Seleccionar | Va a la pantalla | Bajo |

| | | |
|---|---|---|
| | "Selección de operaciones" | |
| Sombrear, checar, el primero (se le preguntaría al cliente) y presionar seleccionar | Selecciona la cuota a pagar | Medio |
| Presionar confirmar | Va a la pantalla de selección de tipo de pago | Bajo |
| Seleccionar efectivo y presionar seleccionar | Va a la pantalla de CONFIRMACION | Bajo |
| Presiono confirmar | Va a la pantalla "Intervención de Documentos" | Bajo |

*TRN11 – Compra de moneda extranjera*

| Realizar acción | Resultado | Think time |
|---|---|---|
| Login (Cajero) | | Bajo |
| Seleccionar el menú "Inicio -> Menú de Caja -> Operaciones de Caja" | Va a la pantalla de Operaciones de Caja | Bajo |
| Seleccionar la transacción 535 - Compra USD | Va a la pantalla de Compra de USD | Bajo |
| Ingresar Monto=10 y presionar "Validar" | Va a la pantalla de confirmación de la información ingresada | Bajo |
| Presionar "Confirmar" | Va a la pantalla de Textos del Ordinal | Bajo |
| Obviar campos y presionar "Confirmar" | Va a la pantalla de "Efectivo en Agencias-Ventanilla" | Bajo |
| Trae datos. Presionar "Continuar" | Va a la pantalla que indica la confirmación de la transacción | Bajo |
| Presionar "Confirmar" | Va a la pantalla de Intervención de Documentos | Bajo |

*TRN12 – Venta de moneda extranjera*

| Realizar acción | Resultado | Think time |
|---|---|---|
| Login (Cajero) | | Bajo |
| Seleccionar el menú "Inicio -> Menú de Caja -> Operaciones de Caja" | Va a la pantalla de Operaciones de Caja | Bajo |
| Seleccionar la transacción 536 - Venta USD | Va a la pantalla de Venta de USD | Bajo |
| Ingresar Monto=10 y presionar "Validar" | Va a la pantalla de confirmación de la información ingresada | Bajo |
| Presionar "Confirmar" | Va a la pantalla de Textos del Ordinal | Bajo |
| Obviar campos y presionar "Confirmar" | Va a la pantalla de "Efectivo en | Bajo |

| | | |
|---|---|---|
| | Agencias-Ventanilla" | |
| Trae datos. Presionar "Continuar" | Va a la pantalla que indica la confirmación de la transacción | Bajo |
| Presionar "Confirmar" | Va a la pantalla de Intervención de Documentos | Bajo |

# 10. Anexo IV - Bitácora

| Fecha (MM/DD/AAAA) | Inicio | Fin | Prueba | Monitorización | Detalles adicionales |
|---|---|---|---|---|---|
| 08/05/2012 | 18:38 | 19:09 | Baseline TRN01 | | Pruebas demoradas debido a que el proceso de cadena de cierre que estaba corriendo no finalizaba completamente y esto falsearía los resultados. Además, el usuario TGGU002 daba error "El cajero está cerrado" al intentar ejecutar una Operación de Caja. Hay algunos datos de los obtenidos para cuentas que no sirven para ejecutar las TRN01 y TRN02. Por ejemplo, la cuenta 465052 es retornada por la consulta que nos enviaron pero no tiene subcuentas asociadas |
| 08/05/2012 | 19:13 | 19:35 | Baseline TRN02 | | Para algunas de las cuentas obtenidas para la TRN02 de Deposito, se saltea el paso de Trabajar con Gastos y esto hace que el script falle. Por ejemplo, para la cuenta 51056 retornada por la consulta. Habría que solicitar cuentas para las cuales se muestre siempre esa pantalla de |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | Trabajar con Gastos |
| 08/05/2012 | 19:50 | 20:16 | Baseline TRN03 | | |
| 08/05/2012 | 20:50 | 21:05 | Baseline TRN04 | | |
| 08/05/2012 | 21:23 | 21:45 | Baseline TRN06 | | La ejecución de ésta transacción falló en el paso seis (no encuentra el texto "CONFIRMACION "). Se verificaron los datos y aparentemente estaban bien, ya que ejecutando de forma manual anduvo para los datos que se probó. Se supone que se rompió el script por alguna razón. De todos modos se ejecutó baseline hasta el paso 5 |
| 08/07/2012 | 05:42 | 05:55 | Baseline TRN11 | | |
| 08/07/2012 | 06:20 | 06:40 | Baseline TRN12 | | |
| 08/08/2012 | 05:41 | 07:10 | 10% de la carga | El Oracle no sintió la carga. En JMX la notificación de errores está limpia | Muchos errores, principalmente de datos. Sólo la TRN03 ejecutó correctamente para la mayoría de las iteraciones |
| 08/08/2012 | 07:36 | 08:40 | 10% de la carga | El Oracle no sintió la carga. En JMX la notificación de errores está limpia | Los tiempos de respuesta aumentaron. Queda la duda si se debe al ingreso de usuarios a trabajar en las oficinas de CNG ya que hubo solapamiento |
| 08/10/2012 | 08:01 | 08:55 | 10% de la carga, extrapolada para las TRN01, TRN02, TRN03, TRN04, TRN11 y TRN12. 180 UV aprox. | nmon en todos los componentes, más el sistema host. JMX para monitorización de la memoria | Se detectó que un programa entraba en *loop*. En el momento de ejecución quedaron al menos 10 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | (heap) de los WAS y CPU utilizada por cada uno. JMX para monitorización de la BD. | requerimientos tirando un número gigantesco de sentencias por segundo. |
| 08/15/2012 | 13:31 | 14:15 | 10% de la carga, extrapolada para las TRN01, TRN02, TRN03, TRN04, TRN11 y TRN12. 180 UV aprox. Un poco menos en realidad para ver si se evitan los problemas que se estaban teniendo en la generadora. Se re-ejecuta porque según nos indican quedó solucionado el problema del programa HCLT0135 que quedaba en loop | nmon en todos los componentes, más el sistema host. JMX para monitorización de la memoria (heap) de los WAS y CPU utilizada por cada uno. JMX para monitorización de la BD. Los logs JMX enviados por CNG presentan solamente 3 minutos de actividad. Se solicita que se envíe la actividad durante toda la prueba. | Sigue pasando lo mismo que reportamos para la TRN01. Para algunos datos particulares el sistema tarda demasiado en responder y luego da el error "Internal Server Error" ya reportado, incluso haciendo la prueba por aplicación se logra reproducir. Un dato particular con el que se da esto es: User:CESCAJ0114 , pwd:123456, cuenta:5580, subcuenta:1. Para esta prueba sucede lo mismo que para la anterior, se corta a la media hora mas o menos y se pierde el acceso a Internet. Conclusión: hay algún problema a nivel de la generadora. OSTA presenta un comportamiento extraño, habilita el botón de Play pero en realidad sigue escribiendo los logs y actualizando las gráficas (aunque los hilos no recopilan tiempos). Esto mismo sucedió para la prueba |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | anterior |
| 08/16/2012 | 06:15 | 06:50 | 10% de la carga, extrapolada para las TRN01, TRN02, TRN03, TRN04, TRN11 y TRN12. 180 UV aprox. Un poco menos en realidad porque según se vio en la prueba anterior al menos duró un poco más la ejecución con esta carga. Se re-ejecuta porque según nos indican ahora sí quedó solucionado el problema del programa HCLT0135 que quedaba en loop | nmon en todos los componentes, más el sistema host. JMX para monitorización de la memoria (*heap*) de los WAS y CPU utilizada por cada uno. JMX para monitorización de la BD | "A nivel de la base de datos la carga fue mínima. A nivel del pool de conexiones no se observaron problemas (siete a ocho conexiones libres de 10), y una carga promedio de 250 sql/seg. Por servidor". De todos modos, resta analizar las estadísticas. |
| 08/22/2012 | 07:40 | 08:30 | 10% de la carga, extrapolada para las TRN01, TRN02, TRN03, TRN04, TRN11 y TRN12. 180 UV aprox. Con *ramp-up* modificado para que sea de 30 minutos para comprobar si se soluciona el problema de las conexiones en OSTA | | Se ejecuta la prueba aunque una proporción relativamente grande de las iteraciones para cualquiera de las transacciones están fallando. Esto probablemente sea a que durante el fin de semana se hizo una migración y se rompieron los datos con los cuales estamos trabajando. La prueba se ejecuta asumiendo que la carga no será representativa (aunque en todos los casos las fallas se dan en el último paso), y principalmente para tener una primer medición del canal. Se posterga la prueba real para cuando los problemas estén solucionados, probablemente el día de mañana. |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | Los resultados muestran que el canal de comunicación con el centro de cómputos está siendo consumido prácticamente completo por la carga generada, por lo que se supone que el mismo es el cuello de botella que hace que los tiempos obtenidos sean altos. Se recomienda cambiar las generadoras físicamente al centro de cómputos para que puedan generar la carga a velocidad LAN. |
| 08/30/2012 | 16:20 | 17:20 | 50% de la carga, extrapolada para las TRN01, TRN02, TRN03, TRN04, TRN11 y TRN12. 180 UV aprox. Escenario OSTA "PRUEBA_50_EXTRAPOLADA" | | |
| 08/30/2012 | 18:22 | 19:22 | 100% de la carga, extrapolada para las TRN01, TRN02, TRN03, TRN04, TRN11 y TRN12. 180 UV aprox. Se ejecuta con *ramp-up* de dos horas y que en teoría demore bastante la prueba. | | |
| 09/18/2012 | 20:21 | 21:41 | 200% de la carga. 3400 UV aprox. TRN01, TRN02, TRN03, TRN04, TRN06, TRN07, TRN07_2, TRN10, TRN11, TRN12. *Ramp-up* de 30 minutos. Ejecución distribuida | | Se para la prueba manualmente luego de 1:20hs de ejecución, ya que los tiempos estaban bastante altos y venían así hacía ya un buen rato. No se registran tiempos ni para la TRN07 ni para la |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | TRN07_2. Para la TRN06 se registran solo unos pocos al inicio. Se nos comenta que durante la prueba había un proceso *batch* ejecutando que luego fue cancelado |
| 09/19/2012 | 20:20 | 21:20 | 100% de la carga. PRUEBA_100_EXTRAPOLADA_V2. 1800 UV aprox. TRN01, TRN02, TRN03, TRN04, TRN06, TRN07, TRN07_2, TRN10, TRN11, TRN12. *Ramp-up* de 30 minutos. Ejecución distribuida | | |
| 09/20/2012 | 20:27 | 21:27 | 150% de la carga. PRUEBA_150_EXTRAPOLADA. 2800 UV aprox. TRN01, TRN02, TRN03, TRN04, TRN06, TRN07, TRN07_2, TRN10, TRN11, TRN12. *Ramp-up* de 30 minutos. Ejecución distribuida | | |
| 09/20/2012 | | | 200% de la carga. PRUEBA_200_EXTRAPOLADA_2. 3600 UV aprox. TRN01, TRN02, TRN03, TRN04, TRN06, TRN07, TRN07_2, TRN10, TRN11, TRN12. *Ramp-up* de 30 minutos. Ejecución distribuida | | Se ejecuta esta carga a solicitud de CNG que deseaba saber qué pasaba con 200%. La carga anterior ya había arrojado tiempos altos. |
| 10/01/2012 | 15:33 | 16:33 | 150% de la carga dirigida hacia el *cluster*. PRUEBA_150_EXTRAPOLADA. 2800 UV aprox. TRN01, TRN02, TRN03, TRN04, TRN06, TRN07, TRN07_2, TRN10, TRN11, TRN12. *Ramp-up* de 30 minutos. Ejecución distribuida | | Repitió lo que ya se había observado, tiempos altos , y carga normal del lado del WAS |
| 10/01/2012 | 16:33 | 17:33 | 150% de la carga directa hacia uno de los WAS. PRUEBA_150_EXTRAPOLADA. 2800 UV aprox. TRN01, TRN02, TRN03, TRN04, TRN06, TRN07, | | A pesar de que el WAS estaba sobrecargado puesto estaba recibiendo la carga de 150% solo, en vez |

| | | | | | |
|---|---|---|---|---|---|
| | | | TRN07_2, TRN10, TRN11, TRN12. *Ramp-up* de 30 minutos. Ejecución distribuida | | balanceada entre dos, resultó tiempos bastante más normales, y estables |
| 10/01/2012 | 17:33 | 18.33 | 150% de la carga directa hacia uno de los nodos IBM HTTP Server. PRUEBA_150_EXTRAPOLADA. 2800 UV aprox. TRN01, TRN02, TRN03, TRN04, TRN06, TRN07, TRN07_2, TRN10, TRN11, TRN12. *Ramp-up* de 30 minutos. Ejecución distribuida | | Repitió los problemas de la primer prueba, pero aumentados, ya que la carga no se balanceaba, sino que era recibida específicamente por un solo IBM HTTP Server |
| 10/03/2012 | 14:57 | 15:57 | 150% de la carga directa hacia el *cluster*. PRUEBA_150_EXTRAPOLADA. 2800 UV aprox. TRN01, TRN02, TRN03, TRN04, TRN06, TRN07, TRN07_2, TRN10, TRN11, TRN12. *Ramp-up* de 30 minutos. Ejecución distribuida | | Se modifican algunos parámetros en las balanceadoras para que soporten más carga. La prueba pretende verificarlo |
| 10/05/2012 | 17:15 | 18:15 | 150% carga total, repartida entre dos generadoras. 75% hacia un WAS, y 75% al otro. TRN01, TRN02, TRN03, TRN04, TRN06, TRN07, TRN07_2, TRN10, TRN11, TRN12. *Ramp-up* de 30 minutos. Ejecución distribuida | | |
| 10/08/2012 | 14:12 | 15:12 | 150% carga total, repartida entre dos generadoras. 75% hacia un WAS, y 75% al otro. TRN01, TRN02, TRN03, TRN04, TRN06, TRN07, TRN07_2, TRN10, TRN11, TRN12. *Ramp-up* de 30 minutos. Ejecución distribuida | | Por pedido de DL&A se reinicia el LPAR y se ejecuta la misma prueba que antes, también apuntando a los WAS 75% y 75%. La distribución fue perfecta, y la carga ejecutada por los WAS fue similar |
| 10/08/2012 | 17:06 | 18:06 | 150% carga dirigida al *cluster*. TRN01, TRN02, TRN03, TRN04, TRN06, TRN07, TRN07_2, TRN10, TRN11, TRN12. *ramp-up* de 30 minutos. Ejecución | | El balanceo no está funcionando bien. Uno de los server (el 26) tiene más carga que el otro (del orden de un 30% |

| | | | | | |
|---|---|---|---|---|---|
| | | | distribuida | | aproximadamente). DL&A solicita a CNG que revea el modo de balance que están realizando los equipos |
| 10/09/2012 | 19:06 | 20:15 | | | Se niveló la memoria de los WAS para que queden los dos en 5 GB. Los WAS comienzan a paginar demasiado. Según personal de DL&A la razón de este proviene de que las balanceadoras están balanceando incorrectamente y además sería deseable asignar más memoria a los WAS |

# 11. Anexo V - Baselines Banco

## TRN01 - Retiro de efectivo en soles

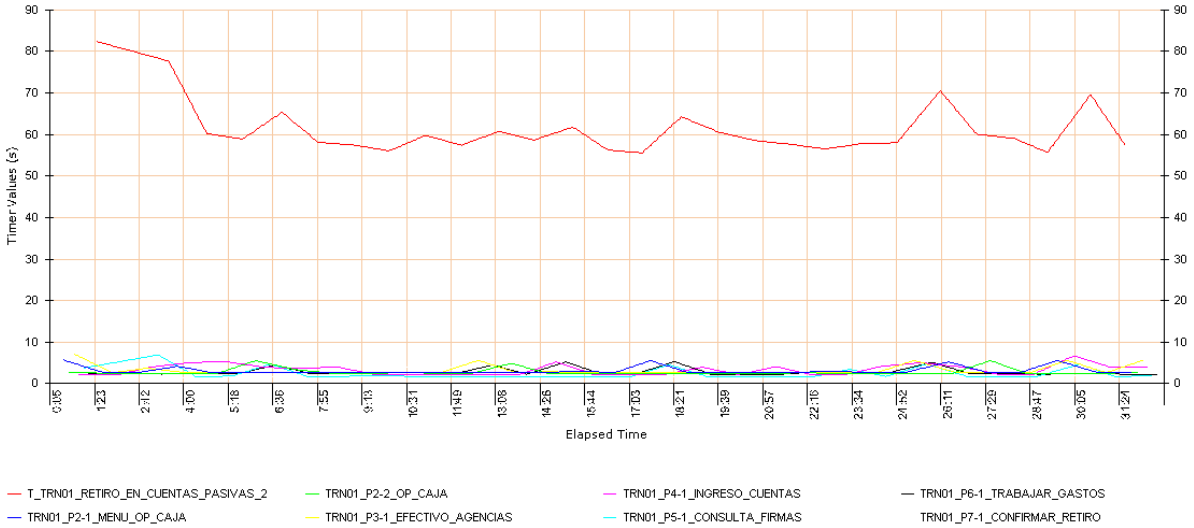
Los tiempos promedio y percentiles obtenidos se pueden visualizar en la *Tabla 6*.

| Nombre paso | Timer | *Promedios (segs)* | *Percentiles 90 (segs)* |
|---|---|---|---|
| 1) Loguearse a la aplicación | TRN01_P1-P-1_ACCESO_PDS | 0.93 | 0.00 |
| 2) Seleccionar el menú Operaciones de caja | TRN01_P2-1_MENU_OP_CAJA | 3.12 | 5.26 |
| 3) Ingresar módulo y transacción y Confirmar | TRN01_P2-2_OP_CAJA | 2.86 | 3.59 |
| 4) Ingresar Importe y Confirmar | TRN01_P3-1_EFECTIVO_AGENCIAS | 3.22 | 5.60 |
| 5) Ingresar número de cuenta, moneda, subcuenta con disponible y Continuar | TRN01_P4-1_INGRESO_CUENTAS | 3.34 | 5.13 |
| 6) Cerrar | TRN01_P5-1_CONSULTA_FIRMAS | 2.47 | 4.70 |
| 7) Cerrar | TRN01_P6-1_TRABAJAR_GASTOS | 2.81 | 4.60 |
| 8) Confirmar | TRN01_P7-1_CONFIRMAR_RETIRO | 2.23 | 4.52 |
| | Transacción completa - Tiempo de procesamiento (sin thinktimes) | 20.97 | 33.41 |
| | Transacción completa baselines (con thinktimes) | 60.97 | 73.41 |
| | Transacción completa carga (con thinktimes) | 85.97 | 98.41 |

| | |
|---|---|
| Thinktimes Baselines - TOTAL | 40 |
| Thinktimes Carga - TOTAL | 65 |

**Tabla 6: Resumen tiempos de respuesta *baseline* TRN01 Retiro de efectivo en soles**

Los valores de las distintas muestras[4] se pueden observar en *Gráfica 116*.




**Gráfica 116: Tiempos de respuesta *baseline* TRN01 Retiro de efectivo en soles**

Se observa que el tiempo total se mantuvo en el entorno de los 60 segundos para la ejecución con los *think times* fijos asignados en valores bajos, por lo que con los *think times* reales resultaría en un tiempo promedio de 85 segundos aproximadamente.

[4] Recordar que la gráfica traza una recta entre los tiempos de respuesta obtenidos en cada muestra (ejecución). El eje horizontal representa el tiempo de prueba en minutos y en el eje vertical el tiempo de respuesta obtenido en segundos.

## TRN02 - Depósito de Efectivo en Soles

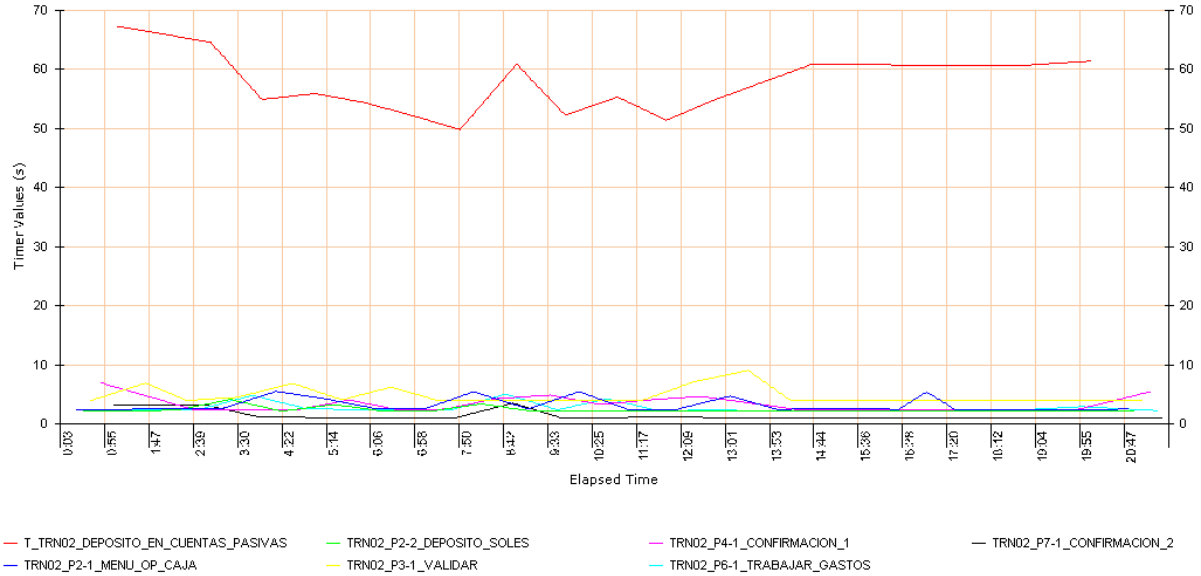
Los tiempos promedio y percentiles obtenidos se pueden visualizar en la *Tabla 7*.

| Nombre paso | Timer | Promedios (segs) | Percentiles 90 (segs) |
|---|---|---|---|
| 1) Loguearse a la aplicación | TRN02_P1-P-1_ACCESO_PDS | 4.10 | 0.00 |
| 2) Seleccionar el menú Operaciones de caja | TRN02_P2-1_MENU_OP_CAJA | 3.17 | 5.26 |
| 3) Seleccionar transacción "501 - Depósito Efectivo Soles S/." | TRN02_P2-2_DEPOSITO_SOLES | 2.43 | 3.59 |
| 4) Ingresar cuenta, suboperación e Importe y Validar | TRN02_P3-1_VALIDAR | 4.88 | 5.60 |
| 5) Confirmar | TRN02_P4-1_CONFIRMACION_1 | 3.52 | 5.13 |
| 6) Cerrar | TRN02_P6-1_TRABAJAR_GASTOS | 2.84 | 4.70 |
| 7) Confirmar | TRN02_P7-1_CONFIRMACION_2 | 1.48 | 4.60 |
| | Transacción completa - Tiempo de procesamiento (sin thinktimes) | 22.43 | 28.89 |
| | Transacción completa baselines (con thinktimes) | 57.43 | 63.89 |
| | Transacción completa carga (con thinktimes) | 82.43 | 88.89 |

| | |
|---|---|
| Thinktimes Baselines - TOTAL | 35 |
| Thinktimes Carga - TOTAL | 60 |

**Tabla 7: Resumen tiempos de respuesta *baseline* TRN02 Depósito de efectivo en soles**

Los valores de las distintas muestras se pueden observar en la *Gráfica 117*.




**Gráfica 117: Tiempos de respuesta *baseline* TRN02 Depósito de efectivo en soles**

Se observa que los tiempos obtenidos en este caso son similares a los obtenidos para la transacción anterior, manteniéndose en el entorno de los 60 segundos considerando los *think times* fijos.

## TRN03 - Alta de DPF (Depósito a Plazo Fijo)

Ver sección “TRN03 - Alta de DPF (Depósito a Plazo Fijo)” del caso en cuestión (“Fusión de dos bancos”).

## TRN04 - Confirmación de Alta de DPF

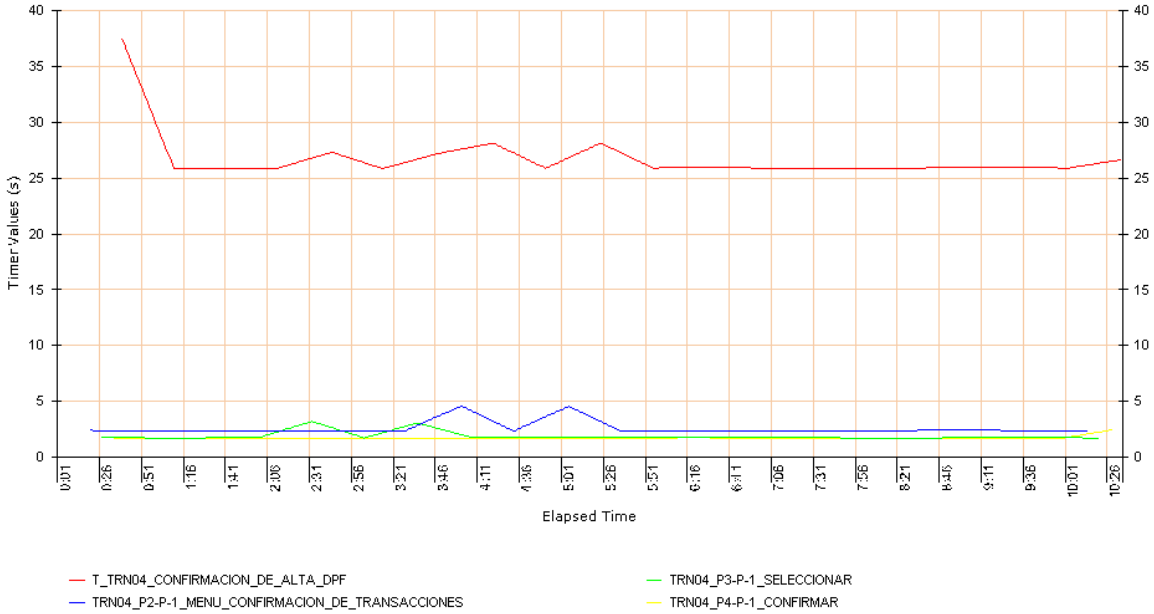
En la *Tabla 8* se detallan los tiempos obtenidos para esta transacción.

| Nombre paso | Timer | Promedios (segs) | Percentiles 90 (segs) |
|---|---|---|---|
| 1) Loguearse a la aplicación | TRN04_P1-P-1_ACCESO_PDS | 0.57 | 0.00 |
| 2) Seleccionar el menú Confirmación de transacciones | TRN04_P2-P-1_MENU_CONFIRMACION_DE_TRANSACCIONES | 2.60 | 2.67 |
| 3) Clickear sobre el primer elemento y Seleccionar | TRN04_P3-P-1_SELECCIONAR | 1.91 | 1.94 |
| 4) Confirar | TRN04_P4-P-1_CONFIRMAR | 1.73 | 1.72 |
| | Transacción completa - Tiempo de procesamiento (sin thinktimes) | 6.80 | 6.33 |
| | Transacción completa baselines (con thinktimes) | 26.80 | 26.33 |
| | Transacción completa carga (con thinktimes) | 51.80 | 51.33 |

| | |
|---|---|
| Thinktimes Baselines - TOTAL | 20 |
| Thinktimes Carga - TOTAL | 45 |

**Tabla 8: Resumen tiempos de respuesta *baseline* TRN04 Confirmación de DPF**

Las distintas muestras de los tiempos obtenidos se pueden observar en la *Gráfica 118*.




**Gráfica** 118**: Tiempos de respuesta *baseline* TRN04 Confirmación de DPF**

Se observa que el tiempo total de la transacción se mantiene estable en el entorno de los 25 segundos durante la ejecución del *baseline*, lo cual equivale a aproximadamente 50 segundos con los *think times* reales.

## TRN11 - Compra en Moneda Extranjera

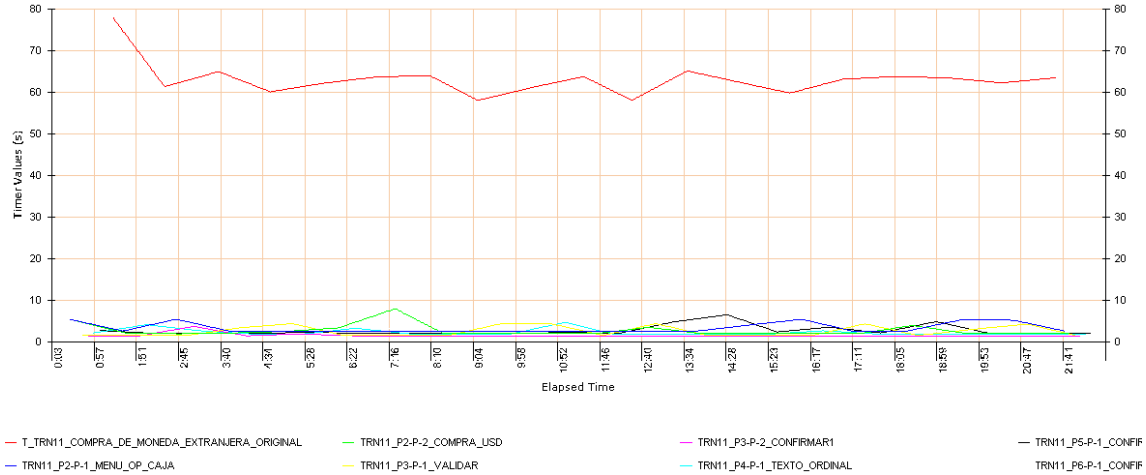
Los resultados *baseline* obtenidos para esta transacción se pueden observar en la *Tabla 9*:

| Nombre paso | Timer | *Promedios (segs)* | *Percentiles 90 (segs)* |
|---|---|---|---|
| 1) Loguearse a la aplicación | TRN11_P1_LOGIN | 0.62 | 0.00 |
| 2) Seleccionar el menú Operaciones de caja | TRN11_P2-P-1_MENU_OP_CAJA | 3.33 | 5.40 |
| 3) Seleccionar transacción "535 - Compra USD" | TRN11_P2-P-2_COMPRA_USD | 2.69 | 4.01 |
| 4) Ingresar monto y Validar | TRN11_P3-P-1_VALIDAR | 2.54 | 4.33 |
| 5) Confirmar | TRN11_P3-P-2_CONFIRMAR1 | 1.54 | 1.52 |
| 6) Confirmar | TRN11_P4-P-1_TEXTO_ORDINAL | 2.29 | 3.34 |
| 7) Continuar | TRN11_P5-P-1_CONFIRMACION | 2.70 | 4.81 |
| 8) Confirmar | TRN11_P6-P-1_CONFIRMAR2 | 2.04 | 2.84 |
| | Transacción completa - Tiempo de procesamiento (sin thinktimes) | 17.74 | 26.24 |
| | Transacción completa baselines (con thinktimes) | 57.74 | 66.24 |
| | Transacción completa carga (con thinktimes) | 57.74 | 66.24 |

| | |
|---|---|
| Thinktimes Baselines - TOTAL | 40 |
| Thinktimes Carga - TOTAL | 40 |

**Tabla** 9**: Resumen tiempos de respuesta *baseline* TRN11 Compra de moneda extranjera**

Las distintas muestras de los tiempos obtenidos en este caso se pueden observar en la *Gráfica 119*.




**Gráfica** 119**: Tiempos de respuesta *baseline* TRN11 Compra de moneda extranjera**

Al igual que para las primeras transacciones los tiempos obtenidos rondan los 60 segundos, tanto considerando los *think times* fijos, reducidos, como los reales, ya que esta transacción tiene la particularidad de que todos los *think times* reales coinciden con los fijos utilizados para la ejecución de *baselines*. Se puede observar que algunos de los pasos para esta transacción superan los 5 segundos.

## TRN12 - Venta en Moneda Extranjera

Se detallan en la Tabla **10** los tiempos correspondientes a la transacción.

| Nombre paso | Timer | *Promedios (segs)* | *Percentiles 90 (segs)* |
|---|---|---|---|
| 1) Loguearse a la aplicación | TRN12_P1_LOGIN | 0.70 | 0.00 |
| 2) Seleccionar el menú Operaciones de caja | TRN12_P2-P-1_MENU_OP_CAJA | 2.20 | 3.49 |
| 3) Seleccionar transacción "536 - Venta USD" | TRN12_P3-P-1_VENTA_USD | 2.02 | 2.70 |
| 4) Ingresar monto y Validar | TRN12_P4-P-1_VALIDAR | 1.65 | 2.58 |
| 5) Confirmar | TRN12_P5-P-1_CONFIRMAR1 | 1.62 | 2.82 |
| 6) Confirmar | TRN12_P6-P-1_TEXTO_ORDINAL | 2.40 | 3.16 |
| 7) Continuar | TRN12_P7-P-1_CONFIRMACION | 2.69 | 3.15 |
| 8) Confirmar | TRN12_P8-P-1_CONFIRMAR2 | 2.21 | 3.87 |
| | Transacción completa - Tiempo de procesamiento (sin thinktimes) | 15.48 | 21.76 |
| | Transacción completa baselines (con thinktimes) | 45.48 | 51.76 |
| | Transacción completa carga (con thinktimes) | 70.48 | 76.76 |

| | |
|---|---|
| Thinktimes Baselines - TOTAL | 30 |
| Thinktimes Carga - TOTAL | 55 |

**Tabla** 10**: Resumen tiempos de respuesta *baseline* TRN12 Venta de moneda extranjera**

# 12. Anexo VI – Guiones Salud

A continuación se muestra la guía de pasos a seguir sobre la aplicación para poder ejecutar las transacciones interactivas. Queda bien definido el flujo a ejecutar y de esta forma bien determinadas las acciones que ejecutarán los robots. En las siguientes tablas se observan las acciones a realizar y el resultado esperado de esta acción, y además se indica el *think time* definido para cada paso. Los *think times* son categorizados en "Alto" = 10 minutos, "Medio" = 5 minutos y "Bajo" = 5 segundos, estos datos surgen de la observación realizada en los dos hospitales "tipo" del uso que le dan los usuarios al sistema.

*TRN01_VISOR_[CONSULTA_ORDENES_SERVICIO]*

| Realizar acción | Resultado | Think time |
|---|---|---|
| **Login (Médico)** | El médico ingresa al sistema | Bajo |
| Clic en buscar | Se despliega el visor | Bajo |

*TRN02_1_FICHAS_DINAMICAS_[INSERTAR_REGISTROS_ESPAÑOL]*

| Realizar acción | Resultado | Think time |
|---|---|---|
| **Login (Médico Español)** | El médico ingresa al sistema | Bajo |
| Clic en el 1er registro que permita "ATENDER" | | Bajo |
| Ingreso al item "HISTORIA DE EMERGENCIA" | Va a la pantalla que carga la ficha | Bajo |
| Modificar (con la flechita hacia arriba) la "HISTORIA DE EMERGENCIA" | Va a la pantalla que permite editar la ficha | Alto |
| Clic en "Grabar" | Va a la pantalla de visualización del item que completé | Medio |
| Clic en "Cerrar" | Se despliega el visor del médico | Bajo |

*TRN02_2_FICHAS_DINAMICAS_[INSERTAR_REGISTROS_PEREIRA]*

Ídem anterior cambia Login con "Medico Pereira" por el "Médico Español"

*TRN03_HISTORIA_CLINICA_[VISUALIZACION]*

| Realizar acción | Resultado | Think time |
|---|---|---|
| **Login (Médico)** | | Bajo |
| Ir a flechita de acciones permitidas. Clic en "VER HISTORIA CLINICA" | Va a la pantalla de "Orden de servicio" | Medio |
| Clic en "Cerrar" | Se despliega el visor del médico | Bajo |

*TRN04_ATENCION_[FINALIZAR]*

| Realizar acción | Resultado | Think time |
|---|---|---|
| **(Médico)** | El médico ingresa al sistema | Bajo |

| | | |
|---|---|---|
| Ir a flechita de acciones permitidas. Clic en "FINALIZAR ATENCIÓN". Elegir que le falte | Va a la pantalla de "Orden de servicio" | Medio |
| Clic en "Cerrar" | Se despliega el visor del médico | Bajo |

*TRN05_BOX_[ASIGNAR]*

| Realizar acción | Resultado | Think time |
|---|---|---|
| **(Triage)** | El triagista ingresa al sistema | Bajo |
| Clic en "INI TRIAGE" del primero | Va a la pantalla de "Respuesta del formulario dinámico" | Medio |
| Ingresar "MOTIVO DE CONSULTA" = "CES_MOTIVO_DE_CONSULTA" y clic en "Grabar" | Se despliega el visualizador de la "FICHA DE TRIAGE" | Bajo |
| Clic en "Confirmar" | Va a la pantalla que permite clasificar | Bajo |
| Se elije clasificación = NIVEL 1 (peor escenario) y clic en "Confirmar" | Se despliega el visor del triage | Bajo |
| Ir a flechita de acciones permitidas. Clic en "ASIGNAR BOX" | Lleva a la pantalla de asignar lugar (hotelería) | Bajo |
| Elegir el 1ero (Apellidos, Nombres = "sin asignar") Y (ESTADO = sin estado o libre) y presionar el botón del medio (asignar) | Lleva a la pantalla que permite confirmar | Bajo |
| Clic en "Confirmar" | Se despliega el visor del triage | Bajo |
| Elegir el 1ero (Apellidos, Nombres = "sin asignar") Y (ESTADO = sin estado o libre) y presionar el botón del medio (asignar) | Lleva a la pantalla que permite confirmar | Bajo |

*TRN06_BOX_[REASIGNAR]*

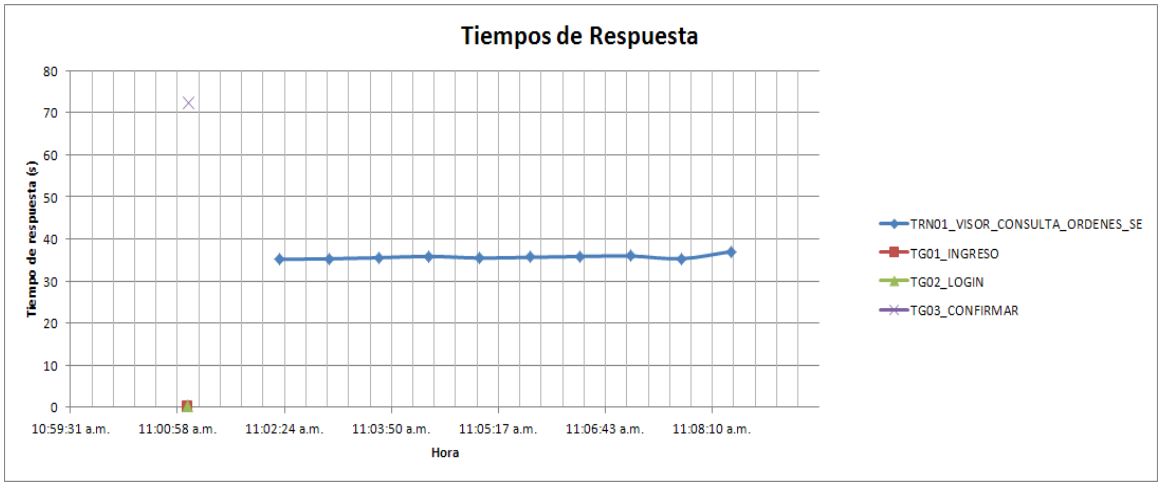
| Realizar acción | Resultado | Think time |
|---|---|---|
| **(Médico)** | El médico ingresa al sistema | Bajo |
| Ir a flechita de acciones permitidas. Clic en "REASIGNAR BOX" | Lleva a la pantalla de asignar lugar (hotelería) | Bajo |
| Elegir el 1ero (Apellidos, Nombres = "sin asignar") Y (ESTADO = sin estado o libre) y presionar el botón de la derecha (reasignar) | Va a la pantalla "Reasignar lugar específico" | Bajo |
| Clic en "Reasignar" | Se despliega el visor del médico | Bajo |

# 13. Anexo VII - Baselines Salud

A continuación se muestran las ejecuciones de los baselines. En la arquitectura planteada por ASSE estaban lejos de ser aceptables. Por esta razón se decidió continuar ejecutando los tiempos base con el objetivo de diagnosticar los posibles problemas, y mejorar la aplicación.
Las pruebas realizadas consistieron en la ejecución secuencial de cada una de las transacciones seleccionadas con 10 iteraciones. De estas pruebas, se obtuvieron indicadores estadísticos de los tiempos de respuesta, tomando en particular, los tiempos máximos, mínimos, promedios y percentiles 90.
A continuación se presentan los resultados de la primera prueba de tiempos base ejecutada en el ambiente propuesto por ASSE.

### TRN01_VISOR_CONSULTA_ORDENES_SERVICIO

| Timers | Promedio | Max | Min | Percentil90 |
|---|---|---|---|---|
| TRN01 | 35,7095 | 37,02 | 35,16 | 36,0435 |
| TG01_INGRESO | 0,086 | 0,086 | 0,086 | 0,086 |
| TG02_LOGIN | 0,282 | 0,282 | 0,282 | 0,282 |
| TG03_CONFIRMAR | 72,423 | 72,423 | 72,423 | 72,423 |



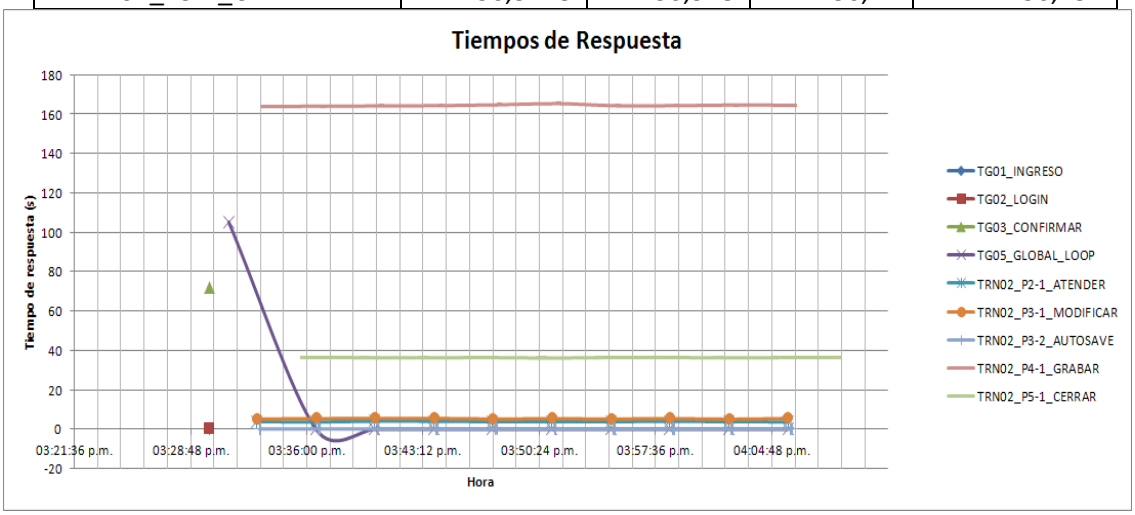

Observando estos resultados, se puede ver que el tiempo de refresque del visor supera siempre los 35 segundos. Esto significa que las operaciones que incluyan la carga del visor, demorarán al menos este tiempo, por lo que por ejemplo cuando el usuario realiza un filtro, deberá esperar 35 segundos para ver los resultados. Esto llevó a que los tiempos no se consideraran aceptables.

Se puede ver también que el tiempo del paso “Confirmar”, es alto, pero cabe destacar que este paso termina en la carga del visor.

### TRN02_1_FICHAS_DINAMICAS_INSERTAR_REGISTROS_ESPANIOL

| Timers | Promedio | Max | Min | Percentil90 |
|---|---|---|---|---|
| TG01_INGRESO | 0,176 | 0,176 | 0,176 | 0,176 |
| TG02_LOGIN | 0,167 | 0,167 | 0,167 | 0,167 |
| TG03_CONFIRMAR | 71,713 | 71,713 | 71,713 | 71,713 |
| TG05_GLOBAL_LOOP | 10,5162 | 105,109 | 0,003 | 10,5199 |
| TRN02_P2-1_ATENDER | 3,8699 | 4,108 | 3,648 | 4,0351 |
| TRN02_P3-1_MODIFICAR | 5,3006 | 5,49 | 5,064 | 5,4783 |

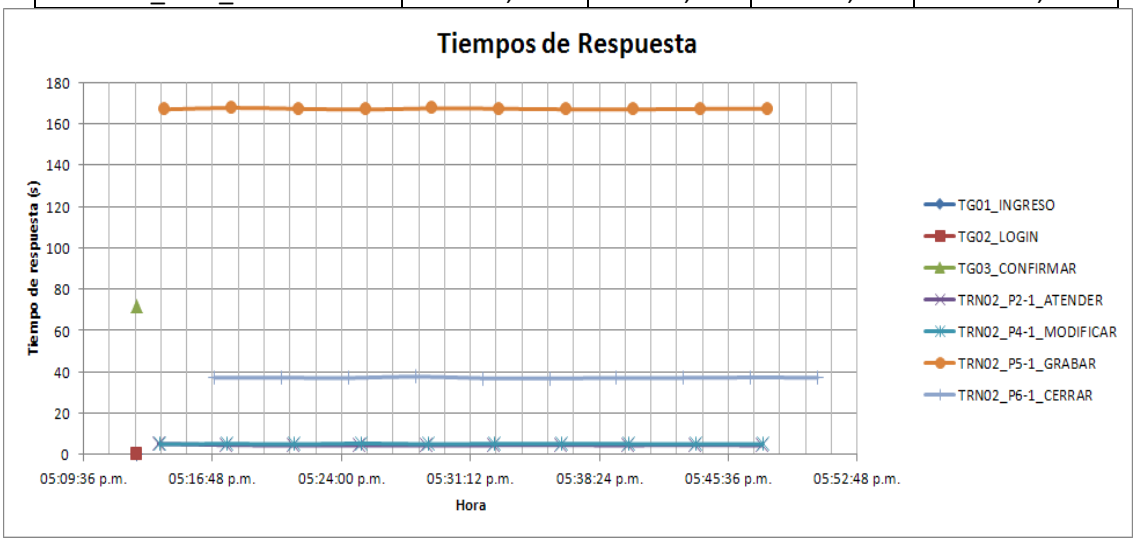
| | | | | |
|---|---|---|---|---|
| TRN02_P3-2_AUTOSAVE | 0,097933 | 0,504 | 0,033 | 0,1424 |
| TRN02_P4-1_GRABAR | 164,3721 | 165,301 | 163,859 | 164,743 |
| TRN02_P5-1_CERRAR | 36,3418 | 36,518 | 36,12 | 36,437 |




Observando los resultados para esta transacción, se puede ver que sobresale el tiempo de grabación. El paso "TRN02_P4-1_GRABAR" supera siempre los 160 segundos (más de 2 minutos y medio), para obtener una respuesta. Esto quiere decir que luego de clickear en grabar, el usuario debe de esperar más de 2 minutos y medios para que se complete la transacción. También, en esta transacción se vuelve a observar, que el paso "TRN02_P5-1_CERRAR", el cual incluye la carga del visor, supera los 35 segundos. Este tiempo se atribuye casi en su totalidad al visor.

## TRN02_2_FICHAS_DINAMICAS_INSERTAR_REGISTROS_PEREIRA

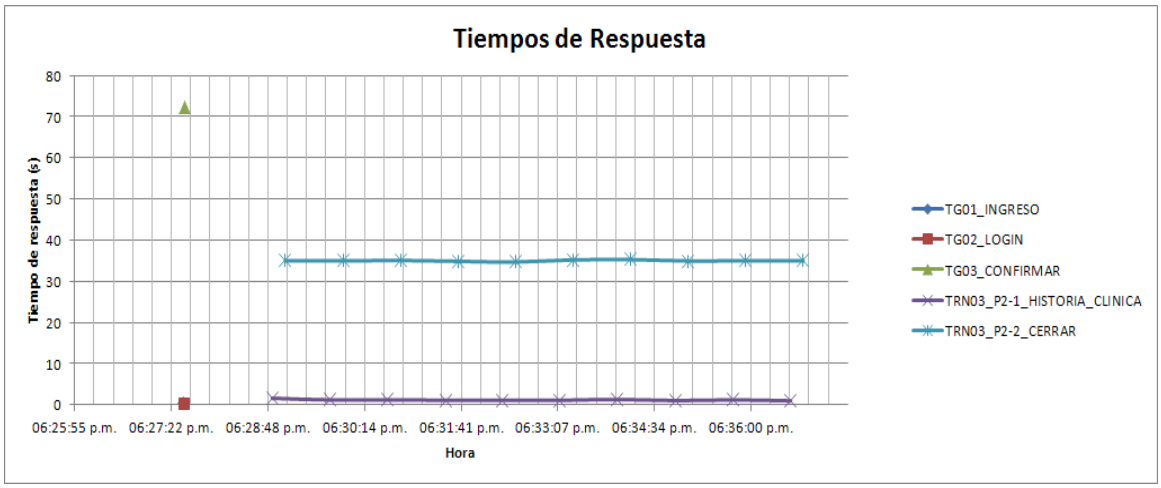
| Timers | Promedio | Max | Min | Percentil90 |
|---|---|---|---|---|
| TG01_INGRESO | 0,297 | 0,297 | 0,297 | 0,297 |
| TG02_LOGIN | 0,356 | 0,356 | 0,356 | 0,356 |
| TG03_CONFIRMAR | 71,653 | 71,653 | 71,653 | 71,653 |
| TRN02_P2-1_ATENDER | 4,594 | 5,213 | 4,363 | 4,7468 |
| TRN02_P4-1_MODIFICAR | 5,1646 | 5,459 | 5 | 5,3123 |
| TRN02_P5-1_GRABAR | 167,1388 | 167,668 | 166,839 | 167,4484 |
| TRN02_P6-1_CERRAR | 37,1836 | 37,751 | 36,829 | 37,3712 |



El comportamiento observado es similar al de la transacción 2_2.

### TRN03_HISTORIA_CLINICA_VISUALIZACION

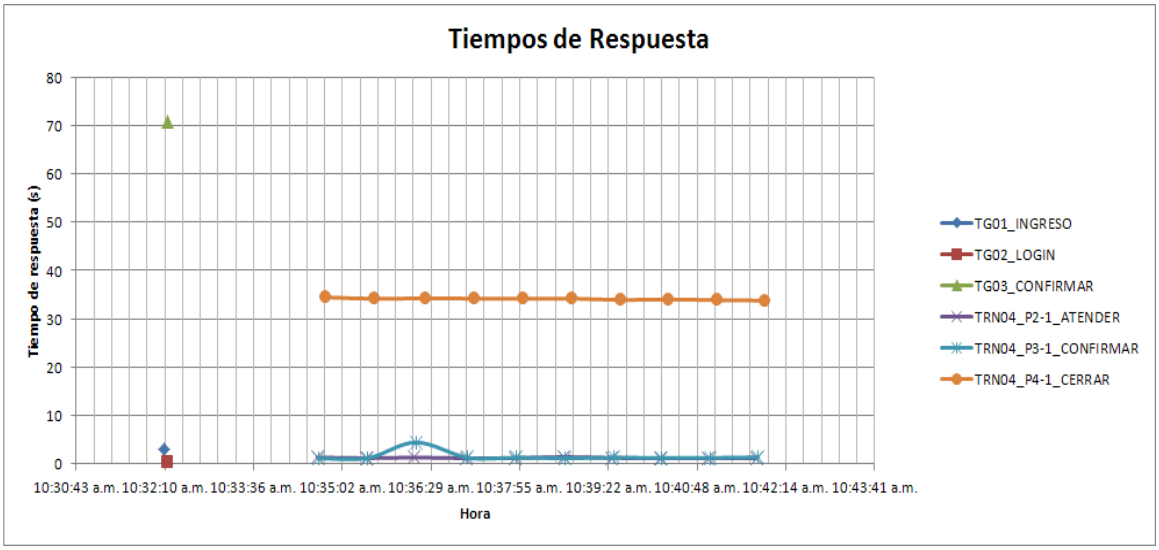
| Timers | Promedio | Max | Min | Percentil90 |
|---|---|---|---|---|
| TG01_INGRESO | 0,386 | 0,386 | 0,386 | 0,386 |
| TG02_LOGIN | 0,173 | 0,173 | 0,173 | 0,173 |
| TG03_CONFIRMAR | 72,274 | 72,274 | 72,274 | 72,274 |
| TRN03_P2-1_HISTORIA_CLINICA | 1,2102 | 1,631 | 1,011 | 1,361 |
| TRN03_P2-2_CERRAR | 35,0139 | 35,294 | 34,708 | 35,2265 |




En este caso, nuevamente el tiempo del paso “Cerrar”, tiene los tiempos más altos. Esto se debe como ya se mencionó porque este incluye la carga del visor.

### TRN04_ATENCION_FINALIZAR

| Timers | Promedio | Max | Min | Percentil90 |
|---|---|---|---|---|
| TG01_INGRESO | 3,111 | 3,111 | 3,111 | 3,111 |
| TG02_LOGIN | 0,474 | 0,474 | 0,474 | 0,474 |
| TG03_CONFIRMAR | 70,704 | 70,704 | 70,704 | 70,704 |
| TRN04_P2-1_ATENDER | 1,2734 | 1,468 | 1,174 | 1,3816 |
| TRN04_P3-1_CONFIRMAR | 1,5975 | 4,477 | 1,164 | 1,714 |
| TRN04_P4-1_CERRAR | 34,1106 | 34,47 | 33,794 | 34,2936 |




Nuevamente se repite el comportamiento de que indica que el tiempo de visor excede lo que sería aceptable.
Debido a problemas en los datos generados, no fue posible realizar las ejecuciones de las transacciones 5 y 6 en esta instancia.

#### 13.1.1.1. Ejecución en escenario alternativo

Como se mencionó anteriormente, las pruebas se realizaron sobre distintas infraestructuras. A continuación se presenta una comparación de los resultados de los tiempos base, en las infraestructuras que se detallan a continuación.

- Servidor de aplicaciones y servidor de base de datos en hosts separados cada uno en una máquina virtual (VMWare).
- Infraestructura propuesta por ASSE- Servidor de aplicaciones y servidor de base de datos en hosts separados cada uno en una máquina virtual (Virtual Box).

En la tabla comparativa 1 y la figura comparativa 1 se pueden comparar los tiempos de ejecución de la transacción 1 en ambas infraestructuras.

TRN01_VISOR_CONSULTA_ORDENES_SERVICIO

| | Promedio VMWare 2 Hosts | Promedio ASSE (s) |
|---|---|---|
| TRN01_VISOR_CONSULTA_ORDENES_SERVICIO | 13,2368 | 33,7102 |
| TG01_INGRESO | 0,124 | 0,233 |
| TG02_LOGIN | 0,095 | 0,264 |
| TG03_CONFIRMAR | 12,488 | 34,762 |

**Tabla Comparativa 1**

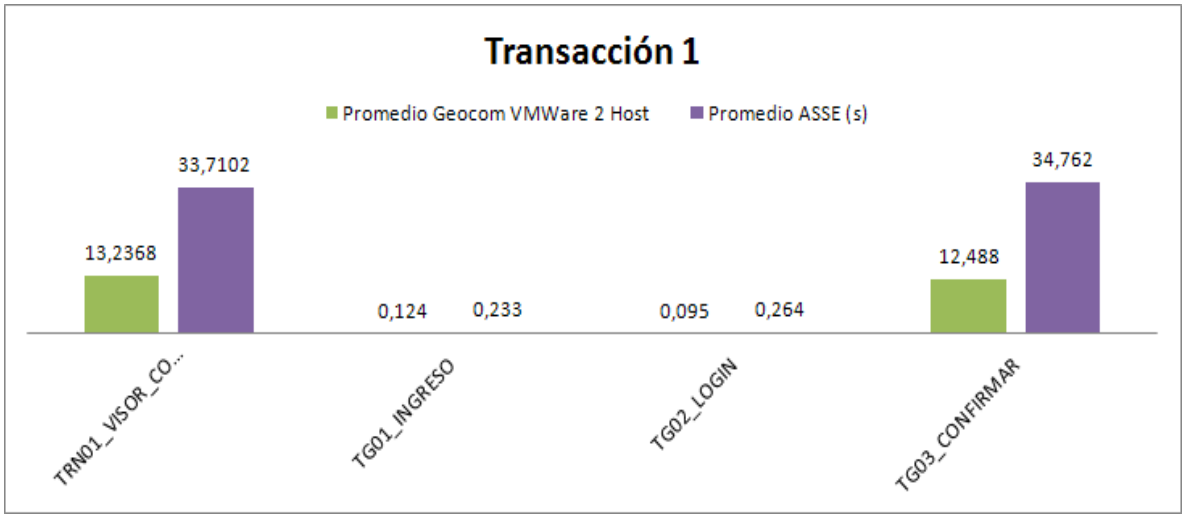



**Figura Comparativa 1**

Es considerable la diferencia de tiempo obtenido entre ambos escenarios. En el caso de la transacción de visualización propiamente dicha (*TRN01_VISOR_CONSULTA_ORDENES_SERVICIO*), se tiene que el tiempo obtenido en ASSE es casi 3 veces mayor que el obtenido en la infraestructura de Geocom.
De todas formas, los tiempos de respuesta del visor para la infraestructura de 2 hosts en geocom, se encuentran fuera del rango aceptable.

### TRN2_1_FICHAS_DINAMICAS_INSERTAR_REGISTRO_ESPANIOL

| | Promedio Geocom VMWare 2 Hosts | Promedio ASSE (s) |
|---|---|---|
| TG01_INGRESO | 0,164 | 0,186 |
| TG02_LOGIN | 0,155 | 0,183 |
| TG03_CONFIRMAR | 13,259 | 34,594 |
| TG05_GLOBAL_LOOP | 3,6272 | 9,9016 |
| TRN02_P2-1_ATENDER | 1,965 | 3,682 |
| TRN02_P3-1_MODIFICAR | 2,4522 | 5,2995 |
| TRN02_P3-2_AUTOSAVE | 0,073033333 | 0,077633333 |
| TRN02_P4-1_GRABAR | 6,1186 | 164,1146 |
| TRN02_P5-1_CERRAR | 13,0022 | 34,2112 |

**Tabla Comparativa 2**

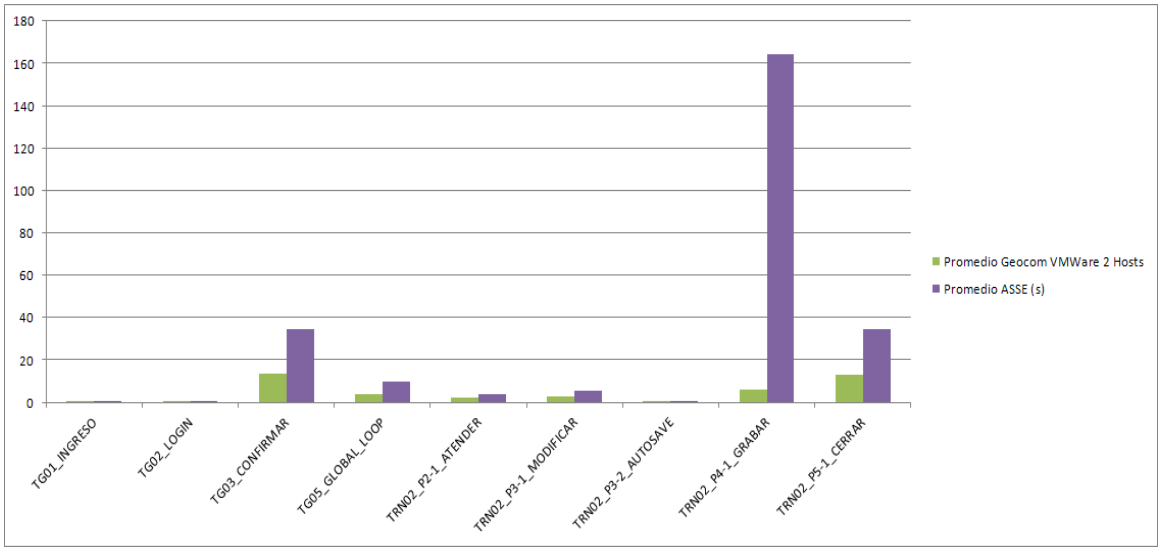



**Figura Comparativa 2**

Esta diferencia replica las diferencias obtenidas en la transacción anterior. Es considerable la diferencia del paso *TRN02_P4-1_GRABAR* la cual ejecuta la lógica que es propia de esta transacción. En el caso de ASSE los tiempos obtenidos no son aceptables bajo ningún concepto (cerca de los 2 minutos 45 segundos), mientras que en el caso de Geocom se encuentra por debajo del segundo.

### TRN2_2_FICHAS_DINAMICAS_INSERTAR_REGISTRO_PEREIRA

| | Promedio Geocom VMWare 2 Hosts | Promedio ASSE (s) |
|---|---|---|
| TG01_INGRESO | 0,077 | 0,157 |
| TG02_LOGIN | 0,15 | 0,19 |
| TG03_CONFIRMAR | 13,536 | 35,365 |
| TRN02_P2-1_ATENDER | 2,7084 | 4,4074 |
| TRN02_P4-1_MODIFICAR | 3,1558 | 4,9914 |
| TRN02_P5-1_GRABAR | 6,4209 | 166,0461 |
| TRN02_P6-1_CERRAR | 15,1145 | 36,5681 |

**Tabla Comparativa 3**

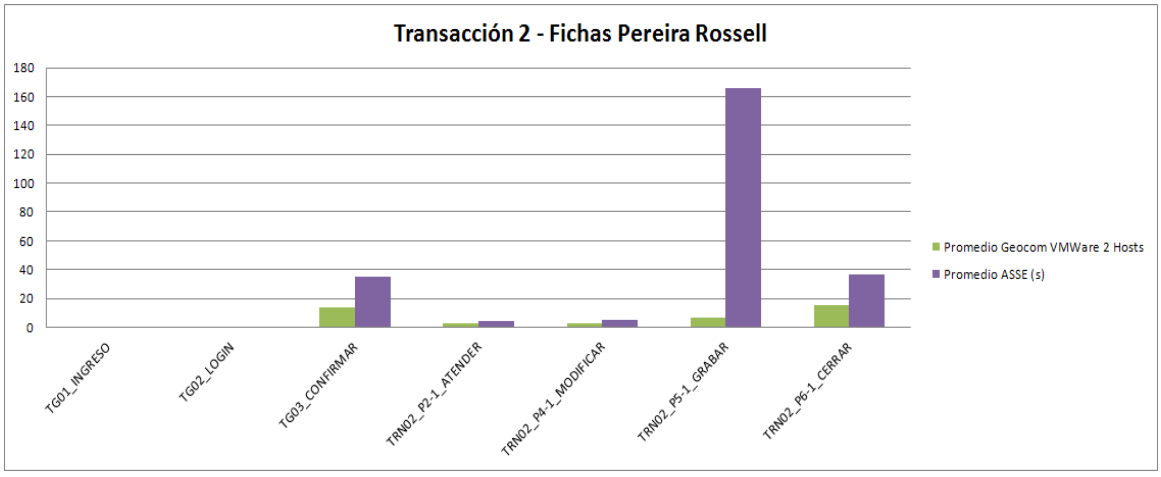



**Figura Comparativa 3**

Los resultados que se pueden ver aquí son similares a los mencionados en las transacciones anteriores.

### TRN3_HISTORIA_CLINICA_VISUALIZACION

| | Promedio Geocom VMWare 2 Hosts | Promedio ASSE (s) |
|---|---|---|
| TG01_INGRESO | 0,075 | 0,193 |
| TG02_LOGIN | 0,111 | 0,182 |
| TG03_CONFIRMAR | 12,303 | 34,526 |
| TRN03_P2-1_HISTORIA_CLINICA | 0,4282 | 1,0345 |
| TRN03_P2-2_CERRAR | 12,499 | 33,039 |

**Tabla Comparativa 4**

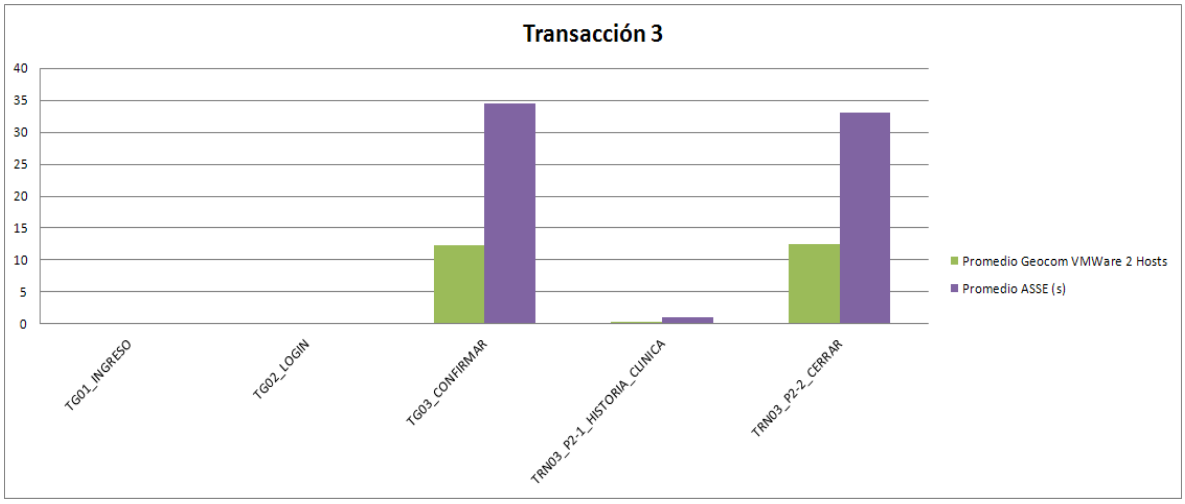



**Figura Comparativa 4**

Los resultados que se pueden ver aquí son similares a los mencionados en las transacciones anteriores.

### TRN4_ATENCION_FINALIZAR

| | Promedio Geocom VMWare 2 Hosts | Promedio ASSE (s) |
|---|---|---|
| TG01_INGRESO | 0,087 | 0,135 |
| TG02_LOGIN | 0,115 | 0,571 |
| TG03_CONFIRMAR | 11,932 | 34,664 |
| TRN04_P2-1_ATENDER | 0,5802 | 1,2571 |
| TRN04_P3-1_CONFIRMAR | 0,6108 | 1,3833 |
| TRN04_P4-1_CERRAR | 12,2733 | 32,6398 |

**Tabla Comparativa 5**

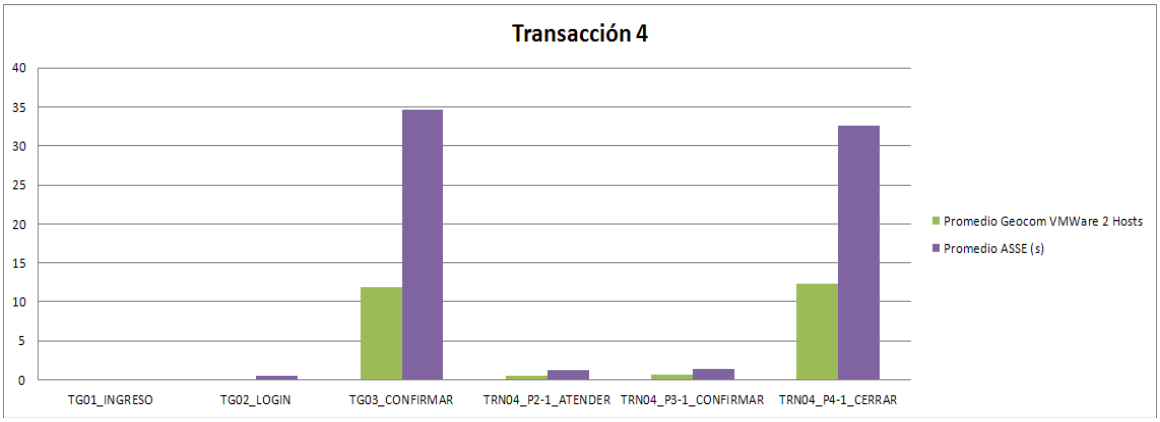



**Figura Comparativa 5**

Nuevamente aquí se confirma el mismo patrón que afecta las transacciones anteriores. Los tiempos del visor son entre 2 y 3 veces mayores en la infraestructura propuesta por Asse.

## Ejecución final de Tiempos Base

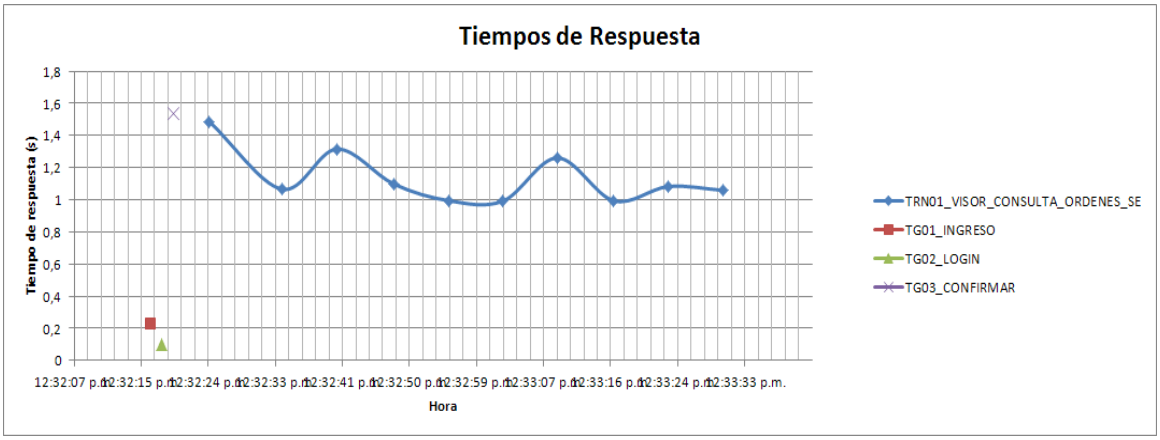
A partir de estas comparaciones, se decidió continuar con las ejecuciones en la infraestructura de Geocom sobre 2 máquinas virtuales en 2 Hosts. Con el objetivo de mejorar los tiempos mejorando la aplicación.

Luego de realizar varios cambios a la aplicación, acompañados de pruebas para cuantificar de esta forma las mejoras y el impacto de las mismas, se llegó a tiempos considerados aceptables para la línea base.

Los resultados de la última medición de tiempos base, tomada el día lunes 18 de febrero fueron los siguientes.

### TRN01_VISOR_CONSULTA_ORDENES_SERVICIO

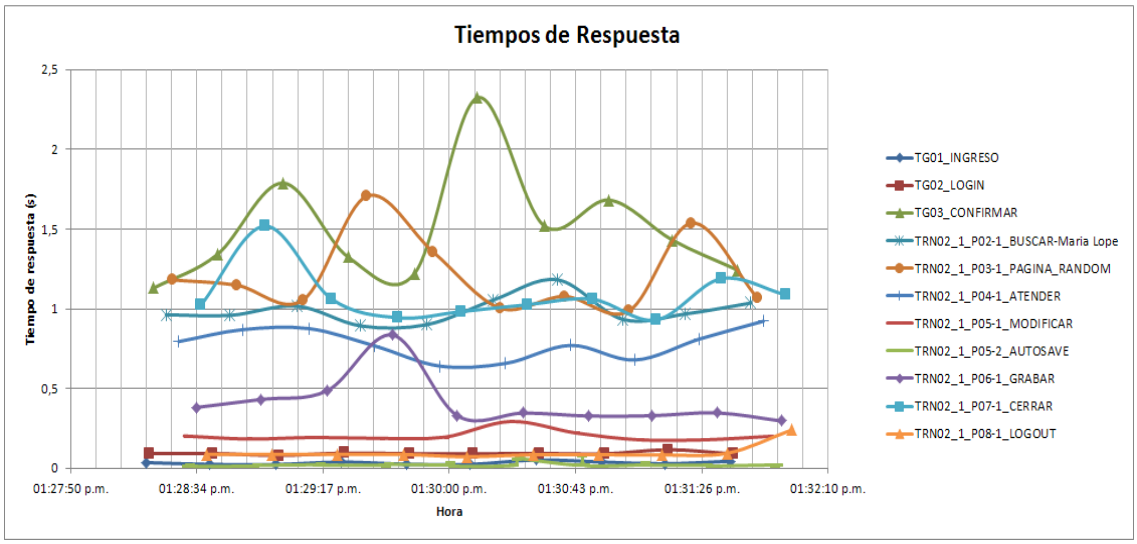
| Timers | Promedio | Max | Min | Percentil90 |
|---|---|---|---|---|
| TRN01_VISOR_CONSULTA_ORDENES_SERVICIO | 1,1354 | 1,488 | 0,993 | 1,3332 |
| TG01_INGRESO | 0,223 | 0,223 | 0,223 | 0,223 |
| TG02_LOGIN | 0,099 | 0,099 | 0,099 | 0,099 |
| TG03_CONFIRMAR | 1,533 | 1,533 | 1,533 | 1,533 |




### TRN02_1_FICHAS_DINAMICAS_INSERTAR_REGISTROS_ESPANIOL

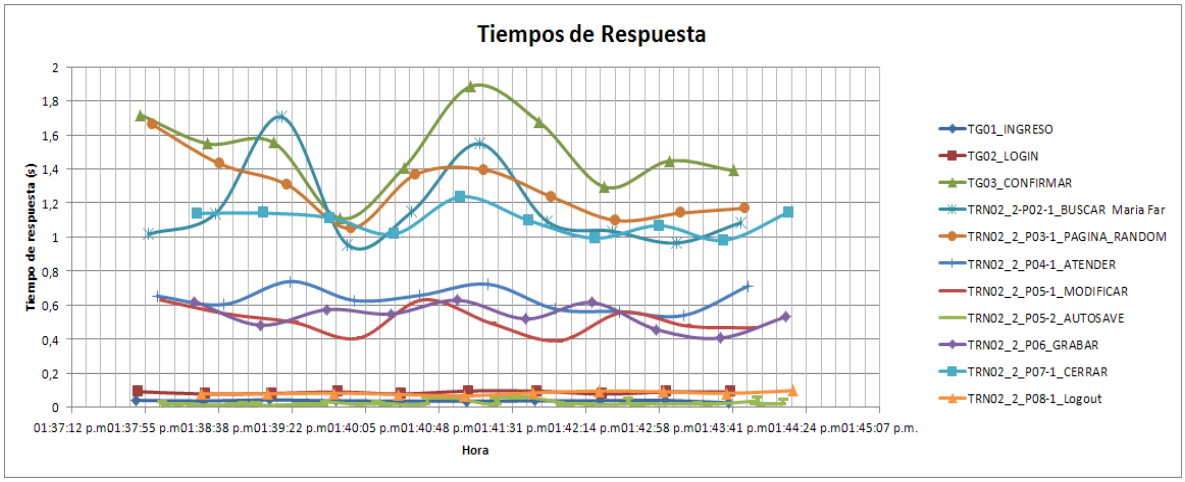
| Timers | Promedio | Max | Min | Percentil90 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| TG01_INGRESO | 0,0343 | 0,053 | 0,024 | 0,0458 |
| TG02_LOGIN | 0,0939 | 0,117 | 0,081 | 0,0981 |
| TG03_CONFIRMAR | 1,5004 | 2,323 | 1,131 | 1,8424 |
| TRN02_1_P02-1_BUSCAR-Maria Lopez | 0,9909 | 1,182 | 0,893 | 1,0677 |
| TRN02_1_P03-1_PAGINA_RANDOM | 1,2126 | 1,707 | 0,987 | 1,5549 |
| TRN02_1_P04-1_ATENDER | 0,779 | 0,923 | 0,641 | 0,8807 |
| TRN02_1_P05-1_MODIFICAR | 0,204 | 0,294 | 0,179 | 0,2292 |
| TRN02_1_P05-2_AUTOSAVE | 0,024267 | 0,079 | 0,013 | 0,0374 |
| TRN02_1_P06-1_GRABAR | 0,4117 | 0,836 | 0,297 | 0,5219 |
| TRN02_1_P07-1_CERRAR | 1,0822 | 1,519 | 0,93 | 1,222 |
| TRN02_1_P08-1_LOGOUT | 0,1013 | 0,241 | 0,073 | 0,1096 |



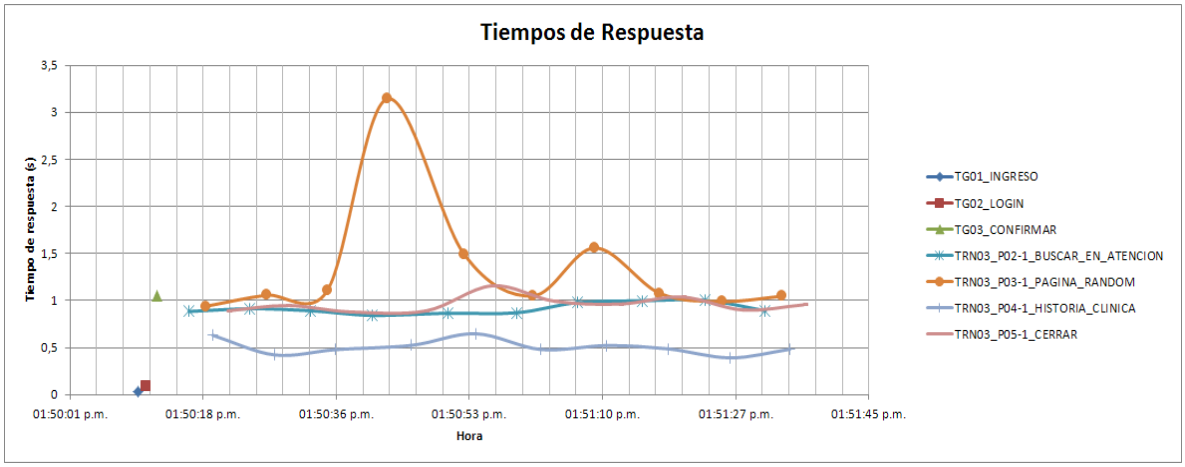

## TRN02_2_FICHAS_DINAMICAS_INSERTAR_REGISTROS_PEREIRA

| Timers | Promedio | Max | Min | Percentil90 |
|---|---|---|---|---|
| TG01_INGRESO | 0,0387 | 0,045 | 0,027 | 0,0423 |
| TG02_LOGIN | 0,0879 | 0,097 | 0,079 | 0,0961 |
| TG03_CONFIRMAR | 1,5032 | 1,886 | 1,11 | 1,7339 |
| TRN02_2-P02-1_BUSCAR  Maria Farias | 1,1693 | 1,706 | 0,956 | 1,5647 |
| TRN02_2_P03-1_PAGINA_RANDOM | 1,2863 | 1,66 | 1,052 | 1,4539 |
| TRN02_2_P04-1_ATENDER | 0,6404 | 0,74 | 0,541 | 0,7256 |
| TRN02_2_P05-1_MODIFICAR | 0,5106 | 0,632 | 0,392 | 0,6311 |
| TRN02_2_P05-2_AUTOSAVE | 0,027872 | 0,069 | 0,013 | 0,059 |
| TRN02_2_P06_GRABAR | 0,5393 | 0,63 | 0,408 | 0,6228 |
| TRN02_2_P07-1_CERRAR | 1,0935 | 1,237 | 0,981 | 1,1551 |
| TRN02_2_P08-1_Logout | 0,0846 | 0,099 | 0,068 | 0,0972 |



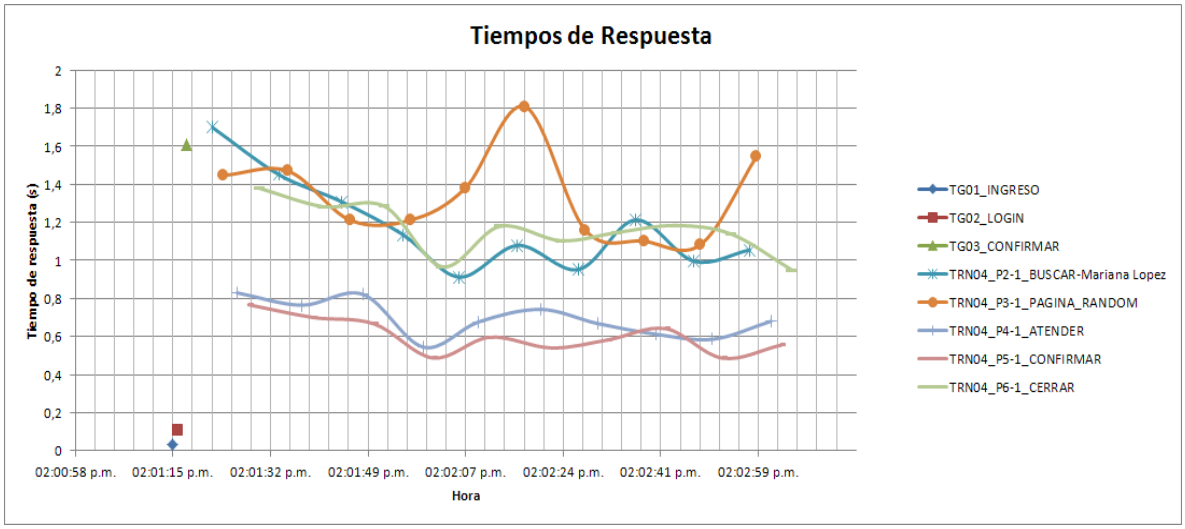

## TRN03_HISTORIA_CLINICA_VISUALIZACION

| Timers | Promedio | Max | Min | Percentil90 |
|---|---|---|---|---|
| TG01_INGRESO | 0,031 | 0,031 | 0,031 | 0,031 |
| TG02_LOGIN | 0,096 | 0,096 | 0,096 | 0,096 |
| TG03_CONFIRMAR | 1,05 | 1,05 | 1,05 | 1,05 |
| TRN03_P02-1_BUSCAR_EN_ATENCION | 0,9148 | 1,003 | 0,844 | 0,9994 |
| TRN03_P03-1_PAGINA_RANDOM | 1,3469 | 3,144 | 0,939 | 1,7184 |
| TRN03_P04-1_HISTORIA_CLINICA | 0,5081 | 0,649 | 0,393 | 0,6337 |
| TRN03_P05-1_CERRAR | 0,9625 | 1,158 | 0,881 | 1,0509 |



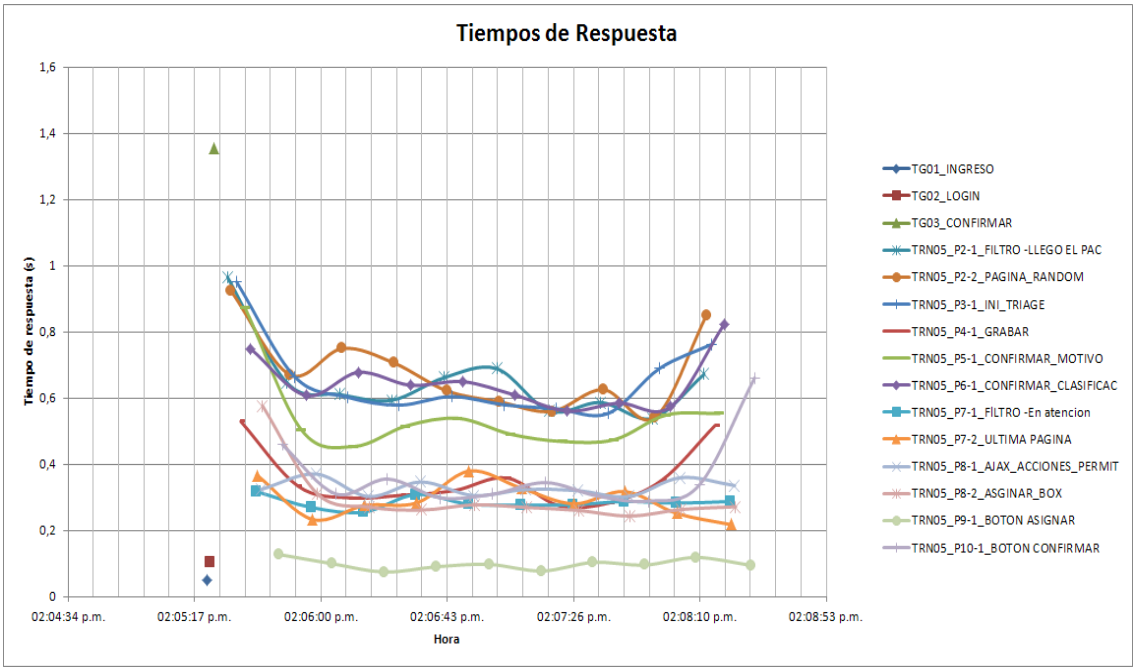

## *TRN04_ATENCION_FINALIZAR*

| Timers | Promedio | Max | Min | Percentil90 |
|---|---|---|---|---|
| TG01_INGRESO | 0,035 | 0,035 | 0,035 | 0,035 |
| TG02_LOGIN | 0,108 | 0,108 | 0,108 | 0,108 |
| TG03_CONFIRMAR | 1,61 | 1,61 | 1,61 | 1,61 |
| TRN04_P2-1_BUSCAR-Mariana Lopez | 1,1801 | 1,698 | 0,913 | 1,4757 |
| TRN04_P3-1_PAGINA_RANDOM | 1,3423 | 1,809 | 1,084 | 1,5732 |
| TRN04_P4-1_ATENDER | 0,6932 | 0,83 | 0,545 | 0,8228 |
| TRN04_P5-1_CONFIRMAR | 0,603 | 0,768 | 0,487 | 0,7077 |
| TRN04_P6-1_CERRAR | 1,1616 | 1,379 | 0,95 | 1,2935 |



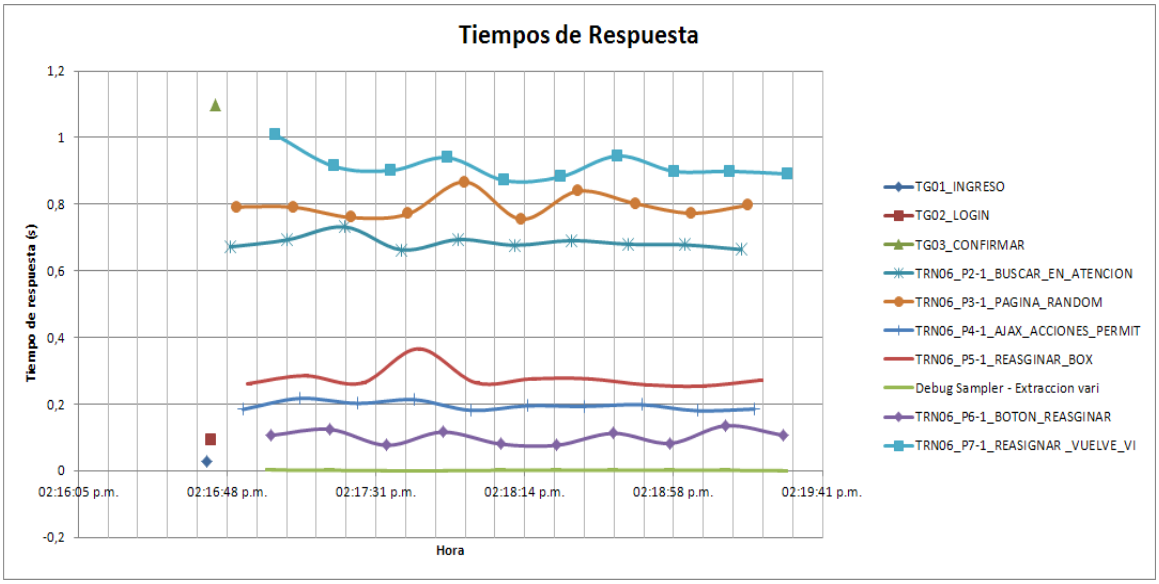

*TRN05_BOX_ASIGNAR*

| Timers | Promedio | Max | Min | Percentil90 |
|---|---|---|---|---|
| TG01_INGRESO | 0,049 | 0,049 | 0,049 | 0,049 |
| TG02_LOGIN | 0,105 | 0,105 | 0,105 | 0,105 |
| TG03_CONFIRMAR | 1,355 | 1,355 | 1,355 | 1,355 |
| TRN05_P2-1_FILTRO -LLEGO EL PACIENTE | 0,6526 | 0,965 | 0,539 | 0,7166 |
| TRN05_P2-2_PAGINA_RANDOM | 0,6839 | 0,924 | 0,541 | 0,8574 |
| TRN05_P3-1_INI_TRIAGE | 0,6557 | 0,951 | 0,554 | 0,7818 |
| TRN05_P4-1_GRABAR | 0,3582 | 0,53 | 0,275 | 0,5192 |
| TRN05_P5-1_CONFIRMAR_MOTIVO | 0,5422 | 0,872 | 0,454 | 0,5858 |
| TRN05_P6-1_CONFIRMAR_CLASIFICACION | 0,6477 | 0,824 | 0,561 | 0,7565 |
| TRN05_P7-1_FILTRO -En atencion triage | 0,2847 | 0,318 | 0,256 | 0,3099 |
| TRN05_P7-2_ULTIMA PAGINA | 0,2937 | 0,38 | 0,218 | 0,3674 |
| TRN05_P8-1_AJAX_ACCIONES_PERMITIDAS | 0,3296 | 0,371 | 0,303 | 0,3611 |
| TRN05_P8-2_ASGINAR_BOX | 0,3011 | 0,577 | 0,244 | 0,3385 |
| TRN05_P9-1_BOTON ASIGNAR | 0,0984 | 0,128 | 0,075 | 0,1199 |
| TRN05_P10-1_BOTON CONFIRMAR | 0,3674 | 0,661 | 0,289 | 0,4792 |




*TRN06_BOX_REASIGNAR*

| Timers | Promedio | Max | Min | Percentil90 |
|---|---|---|---|---|
| TG01_INGRESO | 0,026 | 0,026 | 0,026 | 0,026 |
| TG02_LOGIN | 0,092 | 0,092 | 0,092 | 0,092 |
| TG03_CONFIRMAR | 1,098 | 1,098 | 1,098 | 1,098 |
| TRN06_P2-1_BUSCAR_EN_ATENCION | 0,6844 | 0,732 | 0,663 | 0,6978 |
| TRN06_P3-1_PAGINA_RANDOM | 0,7947 | 0,867 | 0,755 | 0,8418 |
| TRN06_P4-1_AJAX_ACCIONES_PERMITIDAS | 0,1948 | 0,217 | 0,18 | 0,2134 |
| TRN06_P5-1_REASGINAR_BOX | 0,2774 | 0,366 | 0,254 | 0,2931 |
| Debug Sampler - Extraccion variables | 0,0008 | 0,002 | 0 | 0,0011 |
| TRN06_P6-1_BOTON_REASGINAR | 0,1013 | 0,135 | 0,076 | 0,1251 |
| TRN06_P7-1_REASIGNAR _VUELVE_VISOR | 0,915 | 1,008 | 0,871 | 0,9513 |




Los tiempos presentados anteriormente, satisficieron las expectativas para los tiempos base, por lo que se procedió a realizar las ejecuciones incrementales comenzando por la del 20% e incrementando en 20%.

# 14. Bibliografía


A Passive Testing Approach for Protocols in Internet of Things. (2012). *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing.*

ANSI/IEEE. (s.f.). *610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology*. Obtenido de IEEE Standards Assosiation: http://standards.ieee.org/findstds/standard/610.12-1990.html

ANSI/IEEE. (s.f.). *610-1990 - IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. Obtenido de IEEE Standards Assosiation: http://standards.ieee.org/findstds/standard/610-1990.html

Avritzer, A., & Weyuker, E. J. (05 de 1996). Deriving Workloads for Performance Testing. *SOFTWARE PRACTICE AND EXPERIENCE* .

Avritzer, A., & Weyuker, E. J. (1993). Generating Test Suites for Software Load Testing. *Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis.*

Avritzer, A., & Weyuker, E. J. (2005). The role of modeling in the performance testing of e-commerce applications. *IEEE Transactions on Software Engineering (Impact Factor: 1.61).*

Avritzer, A., Bondi, A., & Weyuker, E. J. (2007). Ensuring system performance for cluster and single server systems. *Journal of Systems and Software (Impact Factor: 1.35).*

Avritzer, A., Bondi, A., Grottke, M., & Weyuker, E. J. (2005). Performance Assurance via Software Rejuvenation: Monitoring, Statistics and Algorithms. *2006 International Conference on Dependable Systems and Networks (DSN 2006), 25-28 June 2006, Philadelphia, Pennsylvania, USA, Proceedings.*

Avritzer, A., Kondek, J., Danielle, L., & Weyuker, E. J. (2001). Software performance testing based on workload characterization. *Proceedings of the 3rd international workshop on Software and performance.*

Barbatto, G., & Díaz, N. (2014). *Modelado y ejecución de pruebas de performance.* Montevideo: Facultad de ingeniería.

Barber, R. S. (2006). *PerfTestPlus.* Recuperado el 01 de 2016, de Resources: http://www.perftestplus.com/resources

Bazilinskyy, P., & Brunner, M. (s.f.). Performance Engineering and Testing: The Challenges on Mobile Platforms.

Bourque, P., & Fairley, R. E. (2014). *SWEBOK*. (IEEE Computer Society) Obtenido de IEEE computer society: http://swebok.org/

Bruneo, D., Distefano, S., & Longo, F. (2013). Rejuvenecimiento de Software basado en la carga de trabajo en sistemas Cloud. *IEEE trans. ordenadores: 1072-1085 (2013)* .

Cavalli, A., Lallali, M., Maag, S., Zaidi, F., & Morales, G. (2009). Modeling and Testing of Web-Based Systems.

Cavalli, A., Maag, S., & Morales, G. (2008). Regression and Performance Testing of an e-Learning Web Application: dotLRN. *IEEE Xplore Conference: Signal-Image Technologies and Internet-Based System, 2007. SITIS '07. Third International IEEE Conference.*

*CES*. (s.f.). Recuperado el 01 de 2016, de Centro de Ensayos de Software: http://ces.com.uy

Che, X., & Maag, S. (11 de 2013). *SCIECE CHINA. INFORMATION SCIENCES 53(1):1-18* .

Che, X., & Maag, S. (2013). A Formal Passive Performance Testing Approach for Distributed Communication Systems. *8th International Conference on Evaluation of Novel Software Approaches to Software Engineering* (pág. 11). Angers: ENASE.

Che, X., & Maag, S. (12 de 2014). Formally testing the protocol performances.

Che, X., & Maag, S. (8 de 2014). Passive Performance Testing of Network Protocols. *COMPUTER COMMUNICATIONS 51* .

Che, X., Maag, S., López, J., & Cavalli, A. (2014). Testing Network Protocols: Formally, at Runtime and Online. *The 26th International Conference on Software Engineering and Knowledge Engineering.*
Cotroneo, D., Natella, R., Pietrantuono, R., & Russo, S. (1 de 2014). Una encuesta de estudios de envejecimiento y rejuvenecimiento de software. ACM J. Emerg. Technol Comput. Sist. 10, 1, artículo 8 (enero de 2014), 34 páginas. *ACM J. Emerg. Technol Comput. Sist. 10, 1, artículo 8* .
Hewlett-Packard. (2004). Obtenido de http://sipp.sourceforge.net/
Kaner, C. (11 de 9 de 2012). *The Oracle Problem and the Teaching of Software Testing*. Recuperado el 01 de 2016, de Cem Kaner, J.D., Ph.D. Software Engineering Professor and Consumer Advocate: http://kaner.com/?p=190
Lalanne, F., Che, X., & Maag, S. (2011). Data-centric property formulation for passive testing of communication protocols. *Proceedings of the 13th IASME/WSEAS international conference on Mathematical Methods and Computational Techniques in Electrical Engineering conference on Applied Computing.*
López, J., Che, X., & Maag, S. (2014). An Online Passive Testing Approach for Communication Protocols. *The 9th International Conference on Evaluation of Novel Approaches to Software Engineering.*
Meier, J. D., Farre, C., Bansode, P., & Barber, S. (2007). *Performance Testing Guidance for Web Applications.* Microsoft Corporation.
Microfocus. (2009). *Micro Focus International plc to acquire the Application Testing / AutomatedSoftware Quality business of Compuware Corporation*. Recuperado el 12 de 2015, de Micro Focus International: http://www.microfocus.com/aboutmicrofocus/pressroom/releases/pr20090506457587.asp
Saint-Andre, P. (Core. 2011). *Extensible messaging and presence protocol (xmpp).* IETF RFC 6120.
Smith, C. U., & Williams, L. G. *New Software Performance AntiPatterns: More Ways to Shoot Yourself in the Foot.*
Smith, C. U., & Williams, L. G. (2002). New Software Performance AntiPatterns: More Ways to Shoot Yourself in the Foot.
Statsoft-Dell Software. (s.f.). *Electronic Statistics Textbook*. Recuperado el 03 de 2016, de StatSoft: http://www.statsoft.com/Textbook
Vokolos, F. I., & Weyuker, E. J. (1998). Performance Testing of Software Systems. *Proceedings of the 1st international workshop on Software and performance.*