UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FISICAS Y MATEMATICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACION

VISUAL VERTICAL PROFILING: EVALUAR LA PERFORMANCE Y OPTIMIZAR CAPAS ARQUITECTONICAS

TESIS PARA OPTAR AL GRADO DE
MAGISTER EN TECNOLOGIAS DE LA INFORMACION

CRISTOBAL FELIPE SANFURGO BAUER

PROFESOR GUIA:
ALEXANDRE BERGEL

MIEMBROS DE LA COMISION:
JOSE PINO URTUBIA
PABLO GONZALEZ JURE
LIOUBOV DOMBROVSKAIA

SANTIAGO DE CHILE
2015

RESUMEN

Un problema recurrente en un sistema es la degradación del rendimiento a través del tiempo, las variaciones suelen ser en un principio casi imperceptibles hasta que llega un punto de inflexión donde el aumento de los tiempos de respuesta se hace cada vez mayor. En este punto la identificación y corrección de las desviaciones en el comportamiento esperado del sistema, se vuelve crítico ya que dado el comportamiento exponencial en el aumento de los tiempos de respuesta, nos pone contra el tiempo para evitar el colapso del sistema.

Lo anterior pone en riesgo la continuidad operacional de la organización, la posible disrupción del servicio puede traer consecuencias económicas para la compañía y nuestros clientes, dada esta situación se hace crítico corregir las desviaciones de manera oportuna, rápida y a un bajo costo, sin embargo, no contamos con una forma de identificar que piezas de software son las responsables de los problemas de performance detectados.

Se propone el concepto de Vertical Profiling, la cual es una técnica y metodología que fue definida en la Universidad de Lugano, la cual utilizaremos para generar una representación del comportamiento de una aplicación basado en un set de métricas, donde cada métrica se representa como una serie de tiempo, esto nos permite entender el comportamiento a través del tiempo en cada capa del sistema, independiente de su nivel de abstracción y/o tipo de componente. Esto implica 2 etapas, la primera que consiste en un proceso de recolección de datos a través de todas las capas que componen el software y la segunda etapa que consiste en poder hacer un análisis que considere la variación en el tiempo del comportamiento del sistema. Para poder realizar el análisis del comportamiento, se propone una visualización gráfica, multidimensional, de fácil lectura y oportuna, con el fin de poder identificar los cuellos de botella.

Para este estudio en particular, se utilizará la orientación a aspectos para la captura de datos, y el análisis se realizará utilizando Moose, la cual es una plataforma de análisis de datos, en este caso lo emplearemos para el análisis de Software, y para la visualización utilizaremos Mondrian que nos permite crear el meta-modelo y así realizar un análisis visual del rendimiento del aplicativo. Con la implementación de nuestro Vertical Profiling, se redujeron considerable los tiempos de respuesta de nuestra aplicación de forma rápida, algo que con un profiler tradicional no fue posible.

TABLA DE CONTENIDO

CAPÍTULO 1: INTRODUCCIÓN

Según Roger S. Pressman, ingeniero de software, profesor, consultor y autor de productos centrados en la Ingeniería del Software, la calidad de software es la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados, y con las características implícitas que se espera de todo software desarrollado profesionalmente.

Sin embargo, los sistemas de software enfrentan largas vidas útiles, durante las cuales el entorno de la organización y las condiciones de los requisitos iniciales varían constantemente. Para que tales sistemas respondan a los requisitos operativos para los que fueron desarrollados y que además se ajusten a las nuevas condiciones, se hace necesario mantener el software. Cuando se habla de mantención; generalmente se piensa en nuevas funcionalidades, ya que estas se apalancan con algún nuevo negocio, lo cual genera recursos para dichas actividades, por otro lado, la mantención preventiva no siempre cuenta con los recursos necesarios para poder corregir falencias en el diseño original, que con el paso del tiempo van quedando en evidencia, generando problemas de rendimiento e incluso indisponibilidad del servicio para el cual fue creado el software.

En otras palabras, podríamos decir, que la calidad de un software durante su vida útil puede verse degradada, debido a que el diseño original ahora no tiene concordancia con los requisitos y especificaciones de rendimiento actuales del sistema. Por lo tanto para poder mantener la calidad del software, es necesario hacer un análisis completo y constante de los problemas detectados versus el diseño con el fin de corregir este último para adaptarse a la nueva realidad.

Para poder identificar cuellos de botella en nuestro software, utilizaremos el Vertical Profiling, en particular el enfoque dado por Peter F. Sweeney and Matthias Hauswirth and Amer Diwan, en su publicación “Vertical Profiling: Evaluating Computer Architectures using Commercial Applications”1, en el cual se utiliza el vertical profiling para identificar anomalías en las distintas capas arquitectónicas de un sistema. En este caso nos enfocaremos en las capas del aplicativo, y no en toda la plataforma, adicionalmente agregaremos visualizaciones que nos permitan identificar fácilmente los cuellos de botella y de qué forma corregirlos.

[1] Peter F. Sweeney and Matthias Hauswirth and Amer Diwan. Vertical Profiling: Evaluating Computer Architectures using Commercial Applications [en línea]. Ninth Workshop on Computer Architecture Evaluation using Commercial Workloads, February 2006, Austin, Texas, USA [fecha de consulta: 10 Diciembre 2014]. Disponible en: <http://sape.inf.usi.ch/publications/caecw06>

1.1 CONTEXTO

Hace doce años se creó la primera Plataforma de Comercio Electrónico Chilena con el claro objetivo de dar vida a un servicio que facilitara las relaciones entre el comprador y el proveedor en la industria de la construcción. Fue así como se puso a la industria de la construcción como pionera en la incorporación del e-business en la gestión de los negocios y en las obras del sector Esta iniciativa está hoy plenamente consolidada, sólo el año pasado se transaron US$ 2.161 millones a través de la Plataforma.

Marketplace provee una plataforma electrónica de compra y venta en línea para el sector de la construcción, que abarca desde la identificación de las necesidades de compra hasta el pago a proveedores.

Junto con obtener mejoras en la eficiencia de sus procesos de compra y venta, las empresas asociadas a Marketplace conforman una comunidad de negocios que permite a los proveedores tener una mejor relación con sus clientes e incorporar nuevas empresas a sus carteras, y a los compradores recibir un servicio oportuno y de calidad a lo largo de toda la gestión de abastecimiento.

Dado el crecimiento de esta plataforma, se desarrolló una aplicación de facturación basada en las transacciones mensuales. La facturación de esta plataforma se hace procesando las transacciones hechas por los clientes en la plataforma durante un período mensual. Existen 2 clases de clientes: compradores y proveedores. Para los compradores hay 2 tipos de contratos y para los proveedores 6 tipos de contratos distintos.

El procedimiento para procesar la información es el siguiente:

1. El departamento de operaciones de la plataforma cierra el mes calendario a facturar, esto genera un archivo Excel con las transacciones realizadas en la plataforma durante el periodo con un formato estándar.

2. La aplicación de facturación procesa las transacciones junto al contrato definido para cada empresa, tanto compradora como proveedora y guarda esta información en un formato especial, el cual sirve para exportar para el sistema de contable de la empresa con el fin de general la facturación.

3. La aplicación adicionalmente permite crear y modificar la información de las empresas, tanto la información base, como la del contrato, así como el estado de facturación.

4. La aplicación además permite crear y modificar la información de las facturas generadas, siempre y cuando la facturación está aún activa (nos referimos a facturación activa a la actualmente en curso).

5. Cada factura tiene asociadas las transacciones que dieron origen a esta factura o a ésta.

6. Existe una serie de buscadores tanto para la consulta de facturas como de empresas y sus contratos. Los cuales se ven afectados con una disminución de los tiempos de respuesta a medida que aumenta la carga del sistema. La relevancia de este último punto para el usuario-cliente y por ende para la organización, es la base de la problemática que aborda esta investigación y propuesta del sistema de monitoreo para la optimización del sistema.

1.2 PROBLEMÁTICA

La aplicación de facturación es crítica en la organización ya que es la que permite, primero que todo, cobrar según contrato a los clientes, crear y modificar facturas además de generar una serie de reportes e informes de gestión, fundamentales para poder generar feedback de la plataforma, y mejorar la experiencia de los clientes.

Sin embargo existe una degradación del servicio por el aumento del volumen de datos a procesar. Esta aplicación fue desarrollada hace más de 2 años, en cada periodo de facturación (mensual), se cargan cerca 35.000 registros de transacciones, y se procesan haciendo los cálculos respectivos con el fin de generar la facturación de cada una de las empresas, tanto proveedoras como compradoras, esto genera cerca de 3000 registros en la tabla de facturas. La carga mensual es procesada durante la noche, después del periodo de cierre contable, por lo cual no es un inconveniente dado que el proceso no interfiere con las labores diarias. Sin embargo, diariamente el sistema es utilizado para ingresar nuevas facturas, modificar las generadas automáticamente en el proceso batch, adicionalmente sirve para generar reportes que apoyan la gestión financiera de la empresa, esto último es lo que se requiere mejorar.

Dado que el aumento del volumen de datos ha afectado el rendimiento de la aplicación, se espera mediante el análisis de la aplicación, optimizar el sistema. Los problemas que actualmente se perciben por los usuarios son los siguientes:

- Incremento de los tiempos de respuestas en las consultas y modificaciones de empresas, contratos y facturas.

- Disminución de la experiencia de usuario, con respecto a los tiempos de respuesta, en el uso del sistema, y en la visualización de informes de gestión.

Este sistema ya fue certificado funcionalmente por el área de calidad de la organización, la preocupación ahora se enfoca en los atributos de calidad no funcionales. En este caso nos concentraremos principalmente en los tiempos de respuesta de la aplicación.

La aplicación a analizar está desarrollada sobre tecnología Microsoft, se trata de una aplicación web ASP.NET desarrollada en C#, sobre el Framework 4.0, con una arquitectura N-Layer. Para la capa de persistencia de datos se utilizó el componente Entity Framwork. Sobre un motor de base de datos SQL Server 2008 R2.

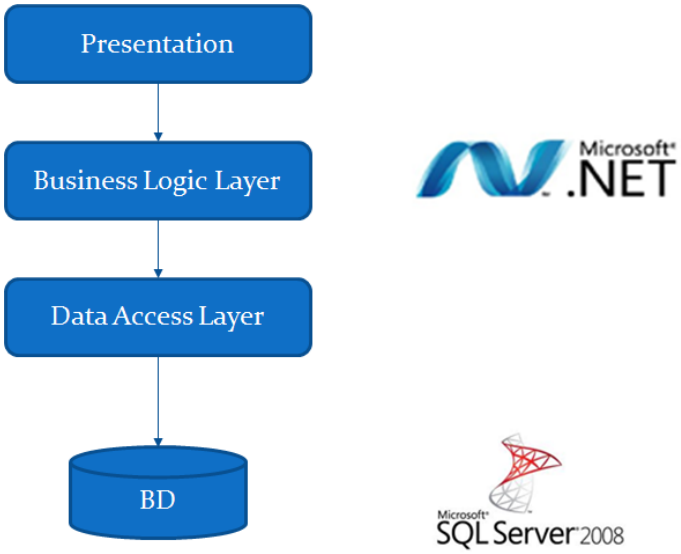


Figura 1 - Modelo arquitectónico "n-Layer" de la aplicación

Con la única métrica que contamos, son los tiempos de los procesos de carga mensual de las facturas, la carga mensual es procesada durante la noche, después del periodo de cierre contable, por lo cual no es un inconveniente dado que el proceso no interfiere con las labores diarias, si bien es cierto, este proceso no es un punto crítico ya que no afecta al uso diario del sistema, nos entrega información relevante para entender la problemática que afecta al sistema. En el cual se puede apreciar que el aumento en los tiempos de carga a través del paso del tiempo, se comporta de forma exponencial, esto nos permite inferir que el resto del sistema tiene el mismo comportamiento, sin embargo no nos da ningún tipo de información sobre donde están el o los cuellos de botella que gatillan este comportamiento.

Gráfico 1 - Benchmark del comportamiento del sistema.

1.3 OBJETIVOS

1.3.1 OBJETIVOS GENERALES

El principal objetivo de este proyecto, es poder anticiparse a posibles dificultades de rendimiento de un software. Y realizar los cambios necesarios para reducir los riesgos de indisponibilidad de servicio y los altos tiempos de respuestas debido al aumento de carga, esto mediante un proceso de mejora continua que permita mantener la calidad del software productivo, considerando las variaciones del entorno de la organización y las condiciones de los requisitos iniciales.

Para lo cual se busca representar gráficamente el resultado de una traza realizada al aplicativo, con esto podremos visualmente hacer un análisis rápido y certero de cuáles son los cuellos de botella que afectan el rendimiento del sistema, con el fin de tener una visualización que nos permita hacer un rápido diagnóstico, a través de un análisis multidimensional.

1.3.2 *OBJETIVOS ESPECÍFICOS*

1. Generar una traza del sistema que nos permita medir nuestro sistema.
    - Metodología: Definir la información que consideraremos relevante para nuestro análisis y posteriormente buscar cómo podemos obtenerla de manera simple, con el menor impacto y costo.
2. Procesar la traza obtenida, con el fin de generar información relevante del comportamiento del sistema.
    - Metodología: A partir de la traza obtenida procesaremos dicha información con el fin de poder identificar cada uno de los aspectos del software que nos interesa analizar.
3. Representar gráficamente el resultado de una traza procesada, que nos permita visualizar de manera simple el comportamiento del sistema.
    - Metodología: Generar un modelo que nos permita representar gráficamente el comportamiento del sistema, según la traza obtenida, con el fin de generar un análisis rápido y certero de cuáles son los cuellos de botella que afectan el rendimiento del sistema, y así tener una visualización que nos permita hacer un rápido diagnóstico, a través de un análisis multidimensional.

4. Optimización y corrección del sistema según el problema detectado.

    - Metodología: Dada la identificación gráfica de los problemas o cuellos de botella detectados, se propondrá e implementará una solución para cada una de estas problemáticas detectadas.

1.4 HIPÓTESIS

Mediante la representación gráfica y multidimensional de un software, podemos ser capaces de resolver problemas intrínsecos del diseño original, el cual no se adapta a las variaciones de entorno de la organización y los cambios de las condiciones de los requisitos iniciales. De una forma rápida y de bajo costo.

1.5 RESULTADOS ESPERADOS

En términos preliminares, se busca que el proyecto cumpla con los siguientes puntos:

- Nos permita tener una visualización coherente del sistema, tanto en su arquitectura de software como su comportamiento.

- Utilizando la visualización nos permita identificar de manera simple y rápida, los problemas de rendimiento.

- Genere una guía de solución a las problemáticas detectadas en la visualización.

1.6 ALCANCES DEL TRABAJO

Para este proyecto utilizaremos un sistema que actualmente está en funcionamiento, esto con el fin de aplicar el modelo. Dado lo anterior y considerando los distintos tipos de software, así como la arquitectura de los mismos, el estudio se basara en dicho software, el sistema de facturación, el cual analizaremos más adelante.

CAPÍTULO 2: METODOLOGÍA

Los problemas de optimización se resuelven usualmente en el área que se denomina "investigación operativa". Este nombre no es casual. La primera parte (investigación) alude al enfoque que debe utilizarse para resolver estos problemas: el de la investigación científica.

2.1 FORMULACIÓN DEL PROBLEMA: ALTOS TIEMPOS DE RESPUESTA DEL APLICATIVO

Dado que el aumento del volumen de datos ha afectado el rendimiento de la aplicación, se espera mediante el análisis de la aplicación, optimizar el sistema.

Los problemas que actualmente se perciben por los usuarios son los siguientes:

- Incremento de los tiempos de respuestas en las consultas, modificaciones de empresas, contratos y facturas.
- Disminución de la experiencia de usuario, con respecto a los tiempos de respuesta en el uso del sistema, y visualización de informes de gestión.

Se hace necesario entonces optimizar el sistema para mejorar los tiempos de respuesta, es en este punto donde nace la pregunta, “¿Qué debemos optimizar/mejorar?”, las respuestas pueden ser variadas, una típica, es agregarle infraestructura (Hardware), específicamente más memoria (RAM) y mayor capacidad de procesamiento (CPU), o bien por otro lado se plantea aplicar reingeniería y modificar el diseño original de la aplicación con el fin de satisfacer los requerimientos de performance, ambas soluciones son por lo general de largo aliento y costosas ya que implica cambios mayores al aplicativo o su infraestructura, ante lo cual sería difícil considerarlas como una respuesta rápida a la problemática, y aun así no podemos garantizar que sea realmente la solución.

Para poder dar una rápida respuesta a nuestro problema asociado a los altos tiempos de respuesta, se buscará identificar de forma rápida y precisa donde están los cuellos de botella de nuestra aplicación, en otras palabras cuales son puntualmente los bloques de código que presentan “problemas de rendimientos”, y qué hacer cuando los identifiquemos.

La siguiente pregunta que debemos responder, es ¿Qué haremos cuando identifiquemos los bloques con “problemas de rendimientos”?, resulta obvia la respuesta, corregirlos. Pero a priori, no tenemos información con respecto a la problemática que debemos resolver, en el peor de los casos, se requerirá rediseñar la solución, lo cual como dijimos con anterioridad no nos permite garantizar una rápida respuesta a la problemática.

Para poder responder con rapidez ante la contingencia, buscaremos soluciones típicas de optimización, y en base a estas soluciones buscaremos los bloques de código

que sean candidatos a utilizar alguna de las soluciones o patrones ya conocidos, dado que podemos estimar el tiempo por optimización a realizar, podemos ir liberando de manera rápida nuevas versiones optimizadas del software.

Las optimizaciones propuestas responden al conocimiento técnico y experiencias anteriores del equipo de desarrollo, dado lo anterior podemos estimar el tiempo de resolución por cada caso, es por esto que buscaremos soluciones que se ajusten a nuestros acotados tiempos.

Soluciones	**Capa (Arquitectura)**	**¿Cuándo?**
Optimizaciones T-SQL	Capa de Datos	Alta tiempos de respuesta Uso recurrente
Manejo de Caché	Capa de Negocio	Uso recurrente Dato invariable
Mejora Interfaz de Usuario	Capa de Presentación	Uso Recurrente

Tabla 1 - Optimizaciones Propuestas

La tabla anterior nos da la información necesaria para poder identificar y priorizar nuestros puntos de mejora. Para poder priorizar es indispensable contar con información del comportamiento del aplicativo; es decir, contar con mediciones cuantitativas de cada uno de los puntos identificados en la Tabla 1, estos puntos los identificaremos como "*Dimensiones*", dado que hacen referencia a un contexto particular de análisis, que por sí sola ya nos entrega información, y en su combinación nos permite además priorizar. De este modo podremos empezar a modificar el código, buscando realizar siempre la modificación en el lugar que represente mayor probabilidad de mejorar los tiempos de respuesta del aplicativo completo.

2.2 RECOLECCIÓN DE LA INFORMACIÓN.

En este paso, definiremos qué información puede representar cada una de las dimensiones que queremos analizar, para lo cual debemos primero que todo, definirlas como tal, a continuación veremos que significa en términos de información requerida del uso del sistema para poder representar dicha dimensión:

- **Dimensión Arquitectura:** Representa el modelo arquitectónico del software, con el fin de individualizar cada pieza de código a analizar, y sus interrelaciones. El objetivo es poder agrupar nuestros bloques de código acorde a la arquitectura de la aplicación.
- **Dimensión Tiempos de Respuesta**: Corresponde a los tiempos de respuesta de cada una de las ejecuciones de las piezas de códigos individualizadas
- **Dimensión Uso:** Corresponde a la cantidad de ejecuciones de cada una de las piezas de códigos individualizadas, para un periodo de tiempo o traza.
- **Dimensión Variabilidad:** Representa a la cantidad de veces que una pieza de código se ejecuta siempre con los mismos parámetros, lo cual nos permite inferir la invariabilidad del resultado.

Es necesario identificar qué datos durante la ejecución del software, nos permitirá identificar cada una de las dimensiones. Para nuestra problemática, es necesario buscar alguna solución de monitoreo. Un profiler es una herramienta para analizar el rendimiento de un programa en tiempo de ejecución, en particular en la frecuencia y duración de las llamadas a métodos. Este puede proporcionar distintas salidas, como una traza de ejecución o un resumen.

Los profilers ofrecen una gran cantidad de herramientas para poder medir e incluso hacer análisis de esta información, claro con cierto costo en los casos de las herramientas de índole comercial, pero lo que nosotros buscamos es dar una rápida respuesta mediante una visualización multidimensional, es decir, en una única visualización poder analizar; donde podemos encontrar nuestro código candidato a ser optimizado. En la tabla siguiente se definen las variables que debemos manejar y a que dimensiones entrega información.

Variable	Significado	Dimensión
Método	Representa el nombre de la pieza de código.	Arquitectura
Parametros	Representa los parámetros con los cuales fue invocado el Método.	Variabilidad
Startime	Nos permite saber cuándo fue tomada la medición. Esto nos permite agrupar según la ejecución registrada.	Uso
Tiempo_ms	Representa el tiempo de ejecución de la pieza de código.	Tiempos de Respuesta
InvocadoPor	Como un método invoca a otros métodos en una distinta capa, debemos obtener la relación entre invocaciones de distintos métodos y capas.	Arquitectura

Tabla 2 - Dimensiones de análisis

Como se mencionó anteriormente, existe muchos profilers y alternativas para el monitoreo de aplicaciones, nosotros utilizaremos la programación orientación a aspectos (POA), también conocida como AOP, por sus siglas en inglés (Aspect-oriented programming), para introducir código con el fin de recabar métricas de las llamadas a métodos y funciones, para esto, primero debemos definir que es un Aspecto. Un Aspecto es una funcionalidad transversal (cross-cutting) que se va a implementar de forma modular y separada del resto del sistema. En nuestro caso el aspecto es el logging (registro de sucesos) dentro del sistema, ya que necesariamente afecta a todas las partes del sistema que generan un suceso, este suceso es la invocación a un método. Por lo tanto podemos recorrer cada método midiendo los tiempos entre un método y otro. Como se describe en la figura siguiente.

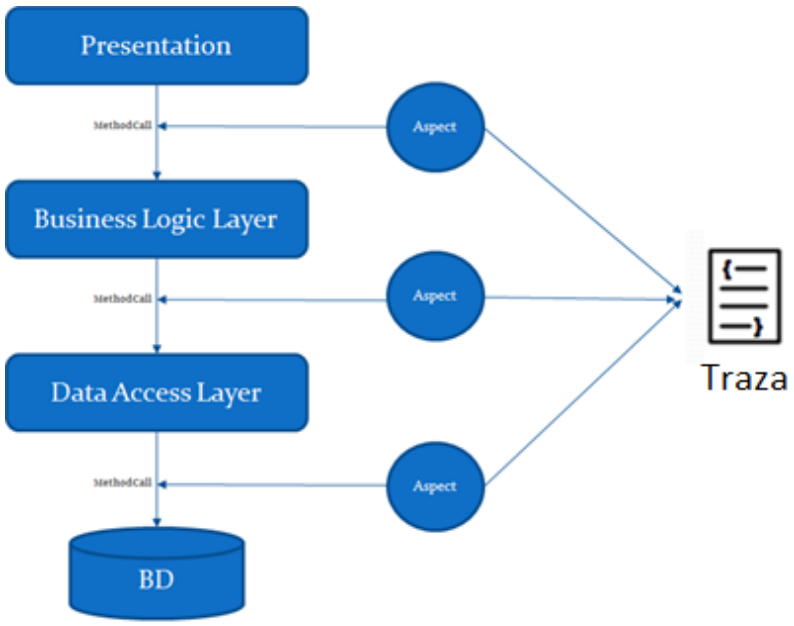


Figura 2 - Utilización de AOP, para crear un profiler personalizado.

Este Aspecto sólo lo utilizaremos en la etapa de certificación, una vez que el aplicativo se encuentre optimizado, este Aspecto de Logging se inhabilitará para evitar la sobrecarga del sistema, gracias a AOP esta inhibición no requiere cambios de código, sólo es una configuración al momento de la compilación del sistema.

Debemos definir la estructura de datos que utilizaremos para nuestra traza, ya que como dijimos necesitamos poder identificar además de los tiempos, el bloque de código y los parámetros, es importante también la interrelación que existe entre los distintos bloques que identificaremos. Para lo cual podemos utilizar una estructura de datos conocida como árbol, para almacenar nuestra traza, con esta estructura de datos podemos identificar las interrelaciones entre los métodos o bloques de código.

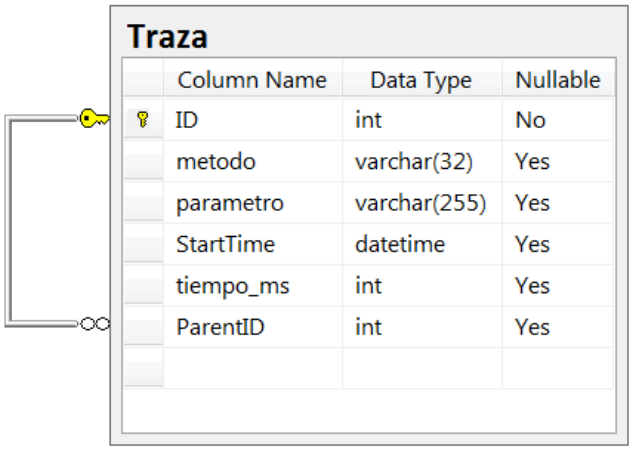


Figura 3 - Estructura traza

En la estructura anterior se define utilizando una base de datos relacional (MS SQL Server), esta tabla que esta auto referenciada, mediante el columna ParentID el cual hace referencia a otra tupla de la misma tabla a través de la columna ID, nos permite así, tener la información almacenada en una estructura de datos del tipo árbol, permitiéndonos así, mantener el orden de llamadas de todas las interacciones del usuario con el sistema, con esto estamos viendo la dimensión de la arquitectura del software. A continuación se presenta el código T-SQL para la creación de esta tabla en el motor de base de datos Microsoft SQL Server 2008R2.

```
--Creamos nuestra tabla trace
CREATE TABLE dbo.traza
(
   ID INT PRIMARY KEY,
   metodo VARCHAR(32),
   parametro varchar(255),
   StartTime datetime,
   tiempo_ms integer,
   ParentID INT FOREIGN KEY REFERENCES dbo.trace(ID)
);
```

Código 1 - Código T-SQL para creación tabla traza

Para continuar con el ejemplo, insertaremos información a la base de datos y luego rescataremos la información.

```
--Insertamos Data de prueba
INSERT INTO traza values(1,'metodoPresentacion1','Parametro=nnn',getdate(),422,NULL)
INSERT INTO traza values(2,'metodoPresentacion2','Parametro=322',getdate(),555,NULL)
INSERT INTO traza values(3, 'metodoNegocio1','Parametro = 33332-2',getdate(),20,2)
INSERT INTO traza values(4, 'metodoNegocio3',' ', getdate(),400 , 1)
INSERT INTO traza values(5, 'MetodoDato','Parametro = 432', getdate(),500 , 4)
INSERT INTO traza values(6, 'metodoPresentacion1','Parametro = hhh',getdate(),300,NULL)

Select ID, metodo, parametro, StartTime, tiempo_ms, ParentID from dbo.traza
```

Código 2 - Código T-SQL para Insertar data a la traza

La siguiente imagen nos muestra el resultado de la consulta anterior, en la cual nos traemos todos los registros de la base de la tabla.

Results

	ID	metodo	parametro	StartTime	tiempo_ms	ParentID
1	1	metodoPresentacion1	Parametro=nnn	2014-11-19 22:13:13.003	422	NULL
2	2	metodoPresentacion2	Parametro=322	2014-11-19 22:13:13.003	555	NULL
3	3	metodoNegocio1	Parametro = 33332-2	2014-11-19 22:13:13.007	20	2
4	4	metodoNegocio3		2014-11-19 22:13:13.007	400	1
5	5	MetodoDato	Parametro = 432	2014-11-19 22:13:13.007	500	4
6	6	metodoPresentacion1	Parametro = hhh	2014-11-19 22:13:13.007	300	NULL

Figura 4 - Valores insertados a la tabla

Con esta información aun nos es difícil revisar la información ya que, si bien es cierto podríamos ordenar por los tiempos de respuesta (tiempo_ms) nos falta obtener mayor información para poder identificar y priorizar cuales son los métodos que podrían ser candidatos a aplicar alguna de nuestras optimizaciones ya identificadas. Para mayor entendimiento y facilidad de lectura, utilizaremos una visualización XML del esquema de datos que a continuación se presenta, se puede ver la cardinalidad además entre los nodos y el tipo de información asociada a cada elemento. En el cual se pueden apreciar de qué forma interactúan las capas y además cabe señalar que según la arquitectura de la aplicación deben ser 3 niveles.

Figura 5 - Diagrama de árbol del Schema XML

Para poder recuperar la información, en la tabla con la traza, con el formato anterior podemos hacerlo utilizando el siguiente código T-SQL.

```
--Le damos formato XML a la traza
SELECT
  -- Capa Presentacion
      Presentacion.metodo,
            Presentacion.parametro,
            Presentacion.StartTime,
  -- Capa Negocio
                        CONVERT(xml, (SELECT
                        Negocio.metodo, Negocio.parametro,Negocio.StartTime,
                                    -- Capa Datos
                                    CONVERT(xml, (SELECT
                                    Dato.metodo, Dato.parametro,Dato.StartTime,
                                    Dato.tiempo_ms
                                    FROM traza Dato Where Dato.ParentID = Negocio.ID
                                    FOR XML PATH ('Capa_Datos')))
                        ,Negocio.tiempo_ms
                        FROM traza Negocio Where Negocio.ParentID = Presentacion.ID
                        FOR XML PATH ('Capa_Negocio')))
            ,Presentacion.tiempo_ms
FROM traza Presentacion Where Presentacion.ParentID is NULL
FOR XML PATH ('Capa_Presentacion'), ELEMENTS, root('Traza')
```

Código 3 - Código T-SQL para exportar traza a XML

```
<Traza>
  <Capa_Presentacion>
    <metodo>metodoPresentacion1</metodo>
    <parametro>Parametro=nnn</parametro>
    <StartTime>2014-11-19T22:13:13.003</StartTime>
    <Capa_Negocio>
      <metodo>metodoNegocio3</metodo>
      <parametro> </parametro>
      <StartTime>2014-11-19T22:13:13.007</StartTime>
      <Capa_Datos>
        <metodo>MetodoDato</metodo>
        <parametro>Parametro = 432</parametro>
        <StartTime>2014-11-19T22:13:13.007</StartTime>
        <tiempo_ms>500</tiempo_ms>
      </Capa_Datos>
      <tiempo_ms>400</tiempo_ms>
    </Capa_Negocio>
    <tiempo_ms>422</tiempo_ms>
  </Capa_Presentacion>
  <Capa_Presentacion>
    <metodo>metodoPresentacion2</metodo>
    <parametro>Parametro=322</parametro>
    <StartTime>2014-11-19T22:13:13.003</StartTime>
    <Capa_Negocio>
      <metodo>metodoNegocio1</metodo>
      <parametro>Parametro = 33332-2</parametro>
      <StartTime>2014-11-19T22:13:13.007</StartTime>
      <tiempo_ms>20</tiempo_ms>
    </Capa_Negocio>
    <tiempo_ms>555</tiempo_ms>
  </Capa_Presentacion>
  <Capa_Presentacion>
    <metodo>metodoPresentacion1</metodo>
    <parametro>Parametro = hhh</parametro>
    <StartTime>2014-11-19T22:13:13.007</StartTime>
    <tiempo_ms>300</tiempo_ms>
  </Capa_Presentacion>
</Traza>
```

Código 4 - Traza en formato XML

2.3 Construcción del modelo: Profiler Multidimensional

El paso siguiente es la construcción de un modelo formal. Se asume que este modelo será una representación suficientemente precisa de las características esenciales del problema, lo que permitirá generar soluciones apropiadas. Pero, en realidad, este supuesto es solo una hipótesis que deberá confirmarse en una etapa posterior, cuando apliquemos el modelo. Al igual que la anterior, esta etapa tampoco es precisamente lineal ni nítidamente separable de la anterior o la siguiente. No hay un único modelo correcto para un problema. El enfoque apropiado es el de la famosa frase de un conocido estadístico (George E. P. Box): "Todos los modelos son erróneos, pero algunos son útiles". Una parte no menor de esta tarea es la de poder ubicar un problema en alguna taxonomía de problemas de optimización, al modo en que un médico diagnostica una enfermedad ubicando los síntomas y signos del paciente.

Definiremos nuestro modelo como un “Profiler Multidimensional”, dado que queremos darle una visualización que nos permita ver distintas dimensiones de manera simultánea, a continuación definiremos las distintas dimensiones de nuestro modelo.

2.3.1 DIMENSIONES

La entrada a nuestro Profiler Multidimensional será nuestra TRAZA, la cual se compone registros interrelacionados y cada uno con los siguientes atributos.

- **Metodo**: Este hace referencia al nombre método invocado.
- **Parametros**: Como su nombre lo indica, se refiere a los parámetros con los cuales se invocó el método.
- **Startime**: Hora de inicio de la invocación. El formato es HH:MM:SS.
- **Tiempo_ms**: Duración en milisegundos de la invocación del método.

Para este análisis no utilizaremos el atributo Startime, solo se tendrá de referencia para poder identificar la fecha de captura de la traza.

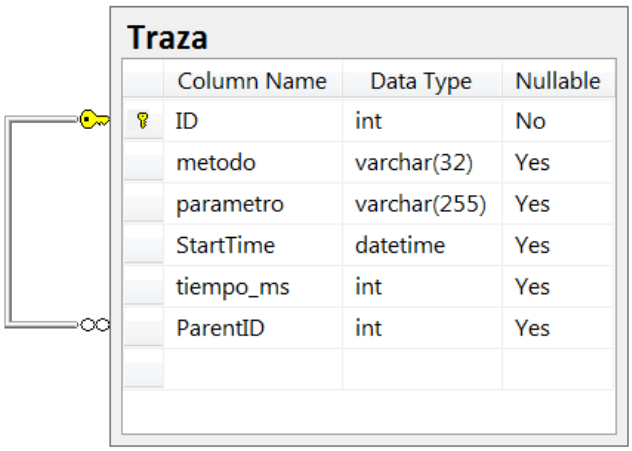


Figura 6 - Estructura traza

Para definir cada una de nuestras dimensiones, utilizaremos el Algebra Relacional [2] con el fin de explicar en qué consiste nuestra dimensión y de qué forma se representará en nuestro Profiler Multidimensional. Adicionalmente para el mejor entendimiento utilizaremos sentencias SQL para reforzar el concepto.

[2] http://es.wikipedia.org/wiki/Algebra_relacional

Utilizaremos la siguiente notación:

Relación	**Proyección**
$RELACION(A_1, A_2, A_{...}, A_n)$	$\pi_{A_x, A_y}(R)$
Join	**Agrupación**
$\boldsymbol{R_1 \ |X|_{A_x} \ R_2}$	$\boldsymbol{G_{A_x}(R)}$

Tabla 3 - Notacion Algebra Lineal

Utilizando algebra relación para definir la TRAZA como una relación. Donde los atributos representan a cada registro de la tabla.

$$TRAZA(metodo, parametros, tiempoms)$$

Ecuación 1 – Relación TRAZA

2.3.1.1. ARQUITECTURA

La primera dimensión que veremos es el de la Arquitectura, para representar la arquitectura de la aplicación N-Layer, como se ve en la “Figura 4. Layout”, utilizaremos un Tree Layout con el fin de representar las distintas capas y entender de qué forma interactúan entre estas. Las líneas que unen a los nodos representan la jerarquía de invocaciones, en cada interacción del usuario con el software.

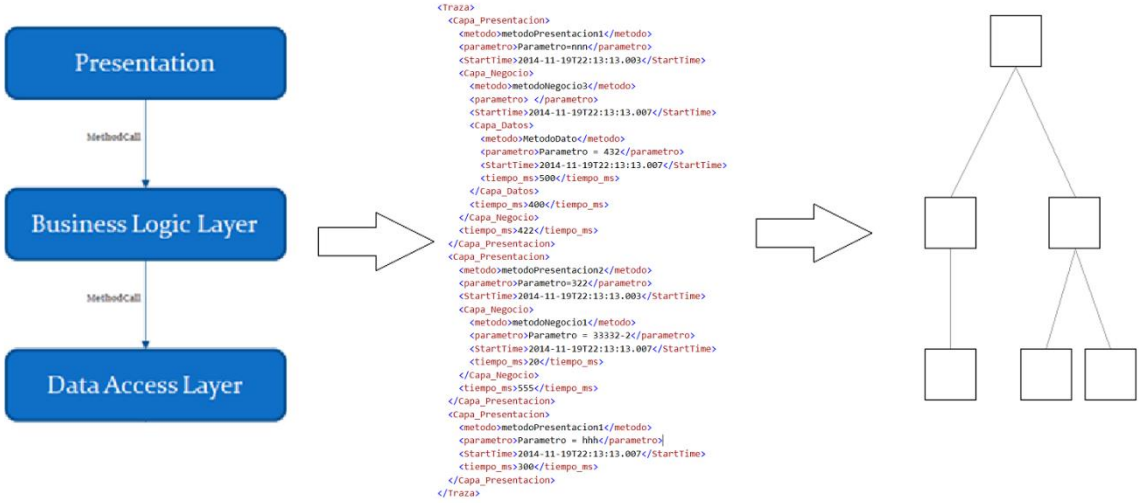


Figura 7 - Layout del profiler Visual (Arquitectura)

De lo anterior se entiende que a partir del uso del sistema, que está representado por el modelo N-Layer, se genera nuestra traza, el cual queremos que se vea reflejado de la misma forma, utilizando el Tree Layout, de este modo podemos identificar las 3 capas. Adicionalmente debemos individualizar el bloque de código, utilizaremos el nombre del método y la capa en que se encuentra.

- **Nodo**

Cada nodo representa a algún método/función/request del sistema, indistintamente de la capa en la cual se invoca y la cantidad de veces invocado, es decir, por cada método/función/request del sistema. Para identificar cada nodo utilizaremos el campo método de la traza, se dibujará en nuestro profiler un objeto, el cual es representado visualmente como un rectángulo.

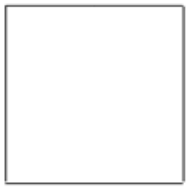


Figura 8- Nodo (método/función/request del sistema)

- **Algebra relacional**

Utilizando algebra relación para definir un nodo como una tupla de la siguiente relación.

$$G_{Distinct(metodo)}TRAZA(metodo)$$

Ecuación 2 –Algebra Relacional, Nodo

Se puede expresar también como la siguiente sentencia SQL

```
SELECT distinct(metodo)
FROM TRAZA
```

Código 5 - T-SQL, Nodo

En esta dimensión tenemos la individualización de los métodos o nodos, y adicionalmente su relación entre ellos, esto último gracias al Tree Layout que utilizaremos ya que este permite mediante las conexiones entre los nodos representar la interacción entre estos. Esto gracias a los distintos modos que se puede recorrer la estructura definida para la traza (tree traversal).

2.3.1.2. Uso

En esta dimensión nos enfocaremos en identificar el uso de cada uno de los nodos, es decir la cantidad de veces que es ejecutado cada bloque de código o nodo como lo definimos anteriormente. Este esta dimensión nos ayudará al momento de priorizar la criticidad de cada nodo.

- **Cantidad de Invocaciones**

La primera métrica que calcularemos al procesar nuestro log, es la cantidad de veces que se invoca cada nodo. Visualmente en nuestro profiler, representará al alto del rectángulo que representa el nodo, a mayor cantidad de invocaciones el alto del nodo aumenta, y viceversa.

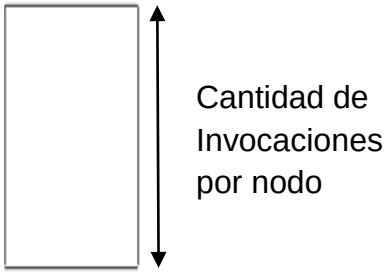


Figura 9 - Cantidad de Invocaciones por nodo

- **Algebra relacional**

Utilizando algebra relación para definir la relación que contiene la información de la cantidad de invocaciones a un mismo método o nodo.

$CANTIDAD(metodo, cantidad)$
$CANTIDAD \coloneqq {}_{metodo}\ G_{count(metodo)}(TRAZA)$

Ecuación 3 –Algebra Relacional, Cantidad de Invocaciones por nodo

Se puede expresar también como la siguiente sentencia SQL

```
SELECT metodo, count(metodo)
FROM TRAZA
GROUP BY metodo
```

Código 6 - T-SQL, Cantidad de ejecuciones por nodo.

2.3.1.3. TIEMPOS DE RESPUESTA

Esta dimensión nos sirve para visualizar en que nodo es donde más tiempo se demora, dado que un nodo o método, al ser invocado con distintos parámetros los tiempos de respuesta pueden variar significativamente, utilizaremos como medida el tiempo de ejecución promedio, para obtener un valor ponderado para nuestro nodo.

- **Tiempo de ejecución promedio**

Utilizaremos el tiempo de ejecución promedio por cada nodo para representar uno de nuestros focos de atención al momento de analizar la situación del aplicativo en nuestro Profiler Multidimensional, esta dimensión estará representada como el ancho del rectángulo que representa el nodo.

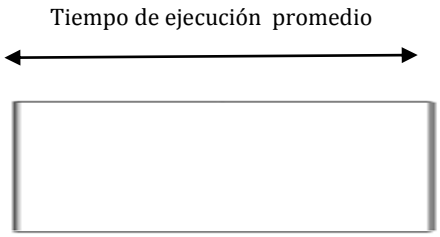


Figura 10 - Tiempo de ejecución promedio por nodo

- **Algebra relacional**

Utilizando Algebra relación para definir la relación que contiene la información del tiempo promedio de cada nodo/método invocado.

$TIEMPO(metodo, tiempoProm)$
$TIEMPO := {}_{metodo}\ G_{avg(tiempoms)}(TRAZA)$

Ecuación 4 –Algebra Relacional, tiempo promedio por nodo

Se puede expresar también como la siguiente sentencia SQL

```
SELECT metodo, avg(tiempoms)
FROM TRAZA
GROUP BY metodo
```

Código 7 - T-SQL, Tiempos de ejecución promedio por nodo.

2.3.1.4. VARIABILIDAD

Esta dimensión nos permitirá reconocer un comportamiento en particular del nodo, que es sobre de qué forma es invocado, nos interesa saber si el método es utilizado para obtener información estática o dinámica. Existe información que no varía constantemente durante el uso diario del aplicativo, como por ejemplo; parámetros de configuración de la aplicación, la información del usuario activo, información de demográfica (comunas, regiones, provincias), entre otros posibles métodos que sean usados para obtener información que no tiene muchas variaciones en el tiempo. Estos métodos son candidatos para hacer uso de caché con el fin de agilizar el acceso a los datos, y reducir cuellos de botellas.

- **Parámetros**

Dado que tenemos un registro de los parámetros con los cuales son invocados nuestros nodos, sacaremos el porcentaje de veces que se repiten los mismos parámetros. Se establece como alta repetición de parámetros para los nodos que son invocados durante la prueba el 100% de las veces con los mismos parámetros, y de baja repetición para los nodos que estén bajo ese valor. Es decir, los nodos rojos son potenciales candidatos a ser optimizados. Se excluyen los métodos que durante la prueba solo se ejecutan una sola vez, dado que esto no representa alta repetición de parámetros, sino que solamente un bajo uso del método.

Figura 11 - Alta repetición de parámetros

Figura 12 - Baja repetición de parámetros

- **Algebra relacional**

Primero debemos calcular cual es la mayor ocurrencia en cuanto a la repetición de los parámetros de entrada al método.

$PARAMETROS(metodo, cantParam)$
$PARAMETROS := \pi_{metodo,\ max(cantidad)}({}_{metodo,\ parametros}G_{count(metodo,parametros)\ cantidad}\ (TRAZA))$

Ecuación 5 - Algebra Relacional, los parámetros más repetidos por nodo

Se puede expresar también como la siguiente sentencia SQL

```
SELECT método, max(cantidad)
FROM (SELECT metodo, parametros, count(*)
        FROM TRAZA
        GROUP by metodo, parametros)
```

Código 8 - T-SQL los parámetros más repetidos por nodo

Y después hacemos un cruce o join con la relación de cantidad de invocaciones por método, con el fin de calcular a que porcentaje corresponde la mayor repetición de parámetros, con esto podemos definir si el método tiene alta o baja repetición de parámetros.

$$REPETICIONPARAM(metodo, porcParam)$$

$$REPETICIONPARAM := \pi_{CANTIDAD.metodo,100*(\frac{Parametro.cantidad}{Cantidad,cantParam})} (PARAMETROS\ |X|_{A_x}\ CANTIDAD\ )$$

Ecuación 6 - Algebra Relacional, el mayor porcentaje de repetición de parámetros por nodo

Se puede expresar también como la siguiente sentencia SQL

```
SELECT  C.metodo,
          100 * (C.cantidad/ P. cantParam)
FROM CANTIDAD C , PARAMETROS P
WHERE E.metodo = P.metodo
```

Código 9 - T-SQL , el mayor porcentaje de repetición de parámetros por nodo

2.4 Análisis Multidimensional: Identificación de puntos de optimización

En este punto debemos verificar que nuestra problemática se vea reflejada efectivamente en una representación formal de nuestro sistema, mediante el modelo definido en la fase anterior, es decir que hemos sido capaces de identificar las taxonomías que estamos buscando.

Soluciones	Capa (Arquitectura)	¿Cuándo?
Optimizaciones T-SQL	Capa de Datos	Alta tiempos de respuesta Uso recurrente
Manejo de Caché	Capa de Negocio	Uso recurrente Dato invariable
Mejora Interfaz de Usuario	Capa de Presentación	Uso Recurrente

Tabla 4 - Optimizaciones Propuestas

En el caso de las optimizaciones T-SQL, gracias a nuestras dimensiones podemos identificar de manera visual el comportamiento buscado. El modelo que se espera obtener busca poder identificar los nodos a los cuales se les debe dar prioridad y adicionalmente que solución implementar. Gracias a la visualización definida en el modelo y de las dimensiones definidas, podemos hacer un análisis, sobre una maqueta del Profiler Multidimensional.

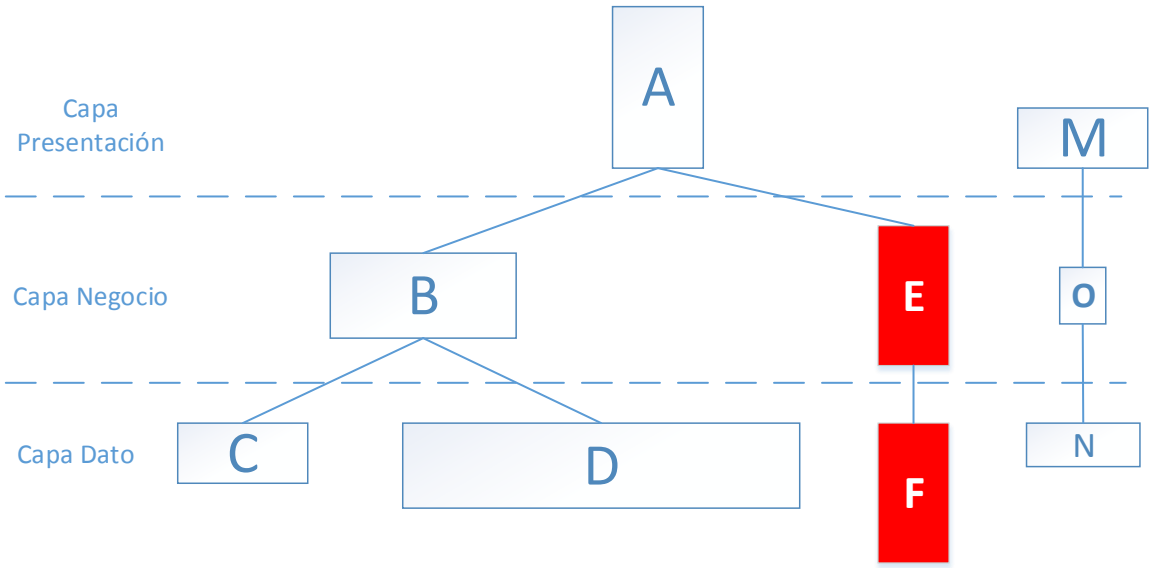


Figura 13 - Maqueta del Profiler Multidimensional.

- **Optimizaciones T-SQL**

Comenzamos por nuestra primera opción de optimización, debemos buscar nuestros candidatos a optimizar las consultas en la base de datos, para esto primero analizamos nuestra arquitectura, y sabemos gracias al Profiler Multidimensional donde está en la capa de datos, con eso descartamos bastantes alternativas, posteriormente nuestro segundo nivel o dimensión de análisis, son los tiempos de respuesta, los cuales están representado por el ancho del rectángulo y la altura del rectángulo también nos indica el uso que tiene dicho método, al menos que este sobre el promedio en cuanto a su uso ya es preocupante. Como se puede ver en el Nodo D, está en la capa de dato, es el más ancho y adicionalmente es el segundo más alto. Es un candidato a optimizar.

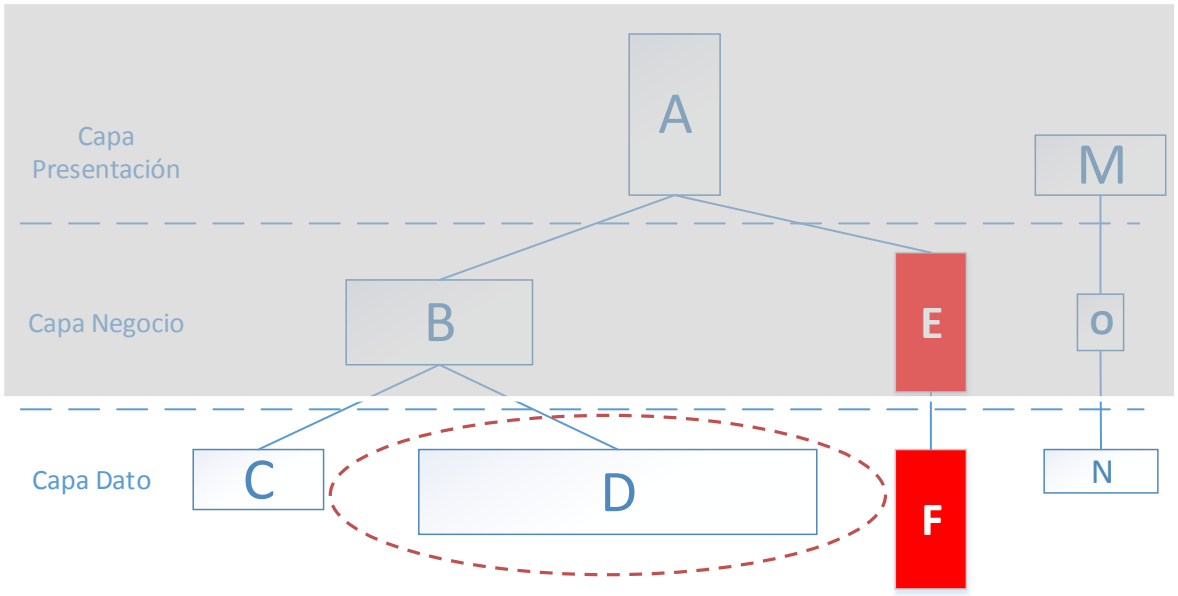


Figura 14 - Análisis Multidimensional Capa Dato

Dimensión	Criterio
Arquitectura	Capa Dato
Tiempos de Respuesta	Mayores tiempos promedio en capa Datos (Ancho)
Uso	Mayor al promedio (Alto)

Tabla 5 - Criterios de selección Nodo Capa de dato

- **Manejo de Caché**

Revisaremos opción de la utilización de caché, debemos buscar nuestros candidatos a optimizar, para esto primero analizamos nuestra arquitectura, y nos enfocamos en la capa de negocio, con eso descartamos bastantes alternativas. Posteriormente nuestro segundo nivel o dimensión de análisis, es el color del nodo, en este caso buscamos los nodos rojos, ya que estos son métodos que están siendo invocados principalmente con los mismos parámetros, esto puede ser indicador de un dato que es invariante durante la ejecución por lo tanto podríamos tenerlo en caché, y con esto disminuir la cantidad de accesos a nuestra base de datos. Como se puede ver en el Nodo E, en la siguiente figura, está en la capa de Negocio, es rojo y adicionalmente es el más alto, eso quiere decir que es el de mayor uso en la capa de negocio.

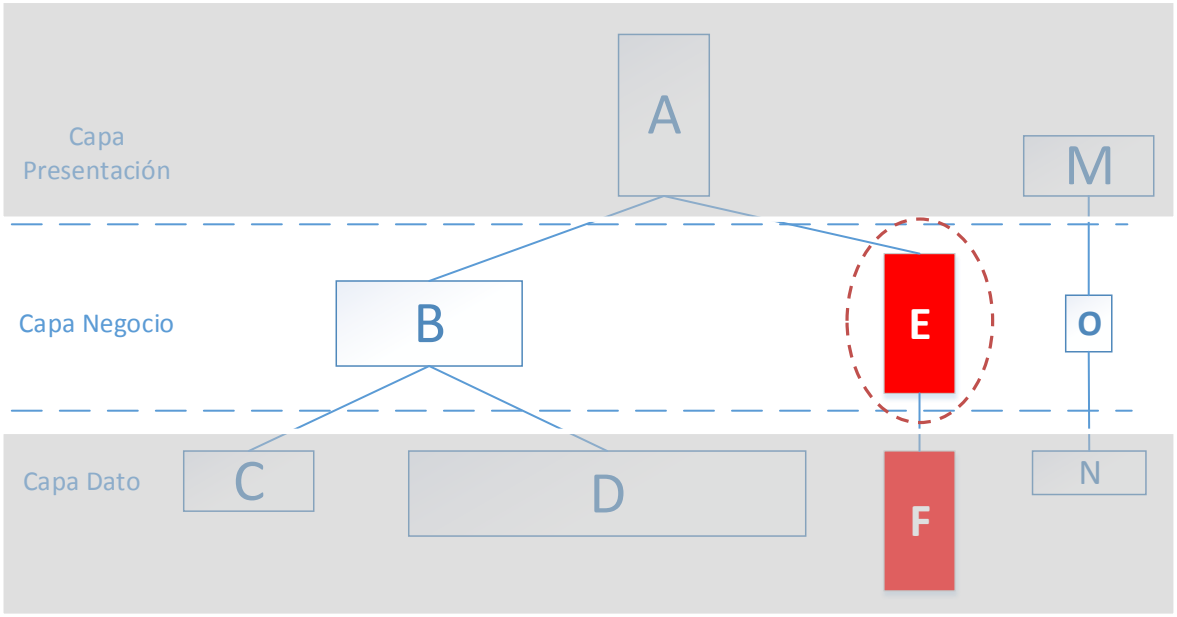


Figura 15 - Análisis Multidimensional Capa Negocio

Dimensión	Criterio
Arquitectura	Capa Negocio
Variabilidad	Dato Invariable (Color Rojo)
Uso	Mayor uso en la capa. (Alto)

Tabla 6 - Criterios de selección Nodo Capa de dato

- **Mejora Interfaz de Usuario**

Revisaremos opción de mejorar la interfaz de usuario, pasándole trabajo al cliente disminuyendo así los request efectuados hacia el servidor, esto puede mejorar además la experiencia del usuario, para esto primero analizamos nuestra arquitectura, y nos enfocamos en la capa de presentación, en nuestro caso es una capa web, posteriormente nuestro segundo nivel o dimensión de análisis es el uso de dicho nodo, se hace relevante la optimización si el uso de esta componente es frecuente, como se puede ver en la figura siguiente, el Nodo A, está en la capa de presentación, es el más alto, eso quiere decir que es la página más usada.

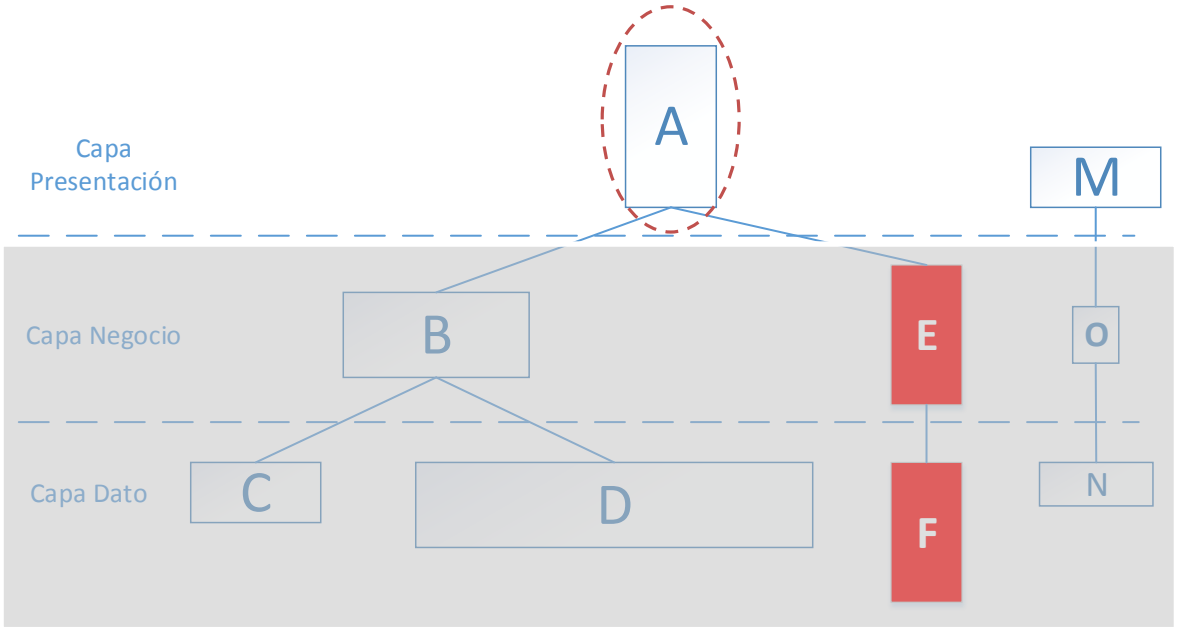


Figura 16 - Análisis Multidimensional Capa Presentación

Dimensión	Criterio
Arquitectura	Capa Presentación
Uso	Mayor uso en la capa. (Alto)

Tabla 7 - Criterios de selección Nodo Capa de Presentación.

CAPÍTULO 3: IMPLEMENTACIÓN

La etapa siguiente es la implementación del modelo. Ya vimos que esta etapa debe estar íntimamente entrelazada con la de modelización. Como la postulación de un modelo es (ni más ni menos) una hipótesis, la forma de comprobarla es mediante experimentación. Mediante los cambios (optimizaciones) realizados al aplicativo y las distintas versiones del software, nos permitirá verificar que el modelo y los cambios realizados a la aplicación cumplan con el objetivo planteado en un principio.

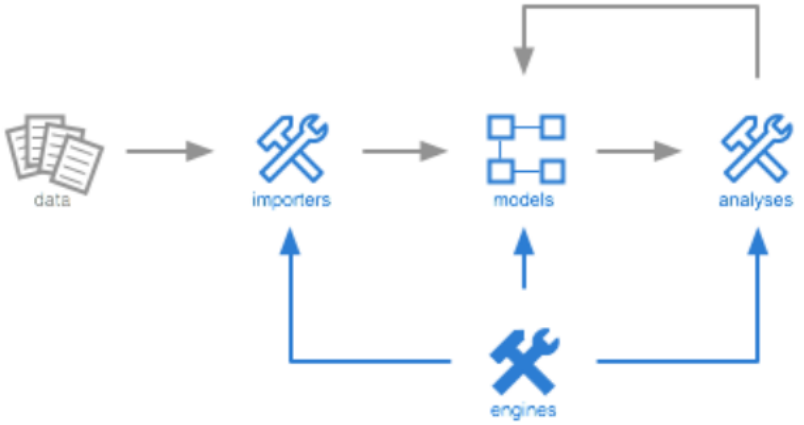
3.1 DISEÑO DE LA PRUEBA

Utilizando la herramienta de MS Visual Studio 2012 para la generación de datos de prueba de carga, se le solicitó al usuario que mediante la herramienta de grabación de uso del sistema, utilizara el sistema de forma normal, durante un periodo de tiempo aproximado de 30 minutos.

Esta herramienta nos permite repetir en cada uno de nuestros escenarios y versiones del software, la misma prueba, con el fin de poder utilizar esto como puntos de evaluación mediante el uso de nuestro profiler multidimensional, obteniendo así para cada ejecución TRAZAS equivalentes en carga y uso de la aplicación.

3.2 PROFILER MULTIDIMENSIONAL

Para poder hacer un análisis de manera certera, oportuna y enfocada en los reales problemas de rendimiento, utilizaremos *Moose*[3], la cual es un plataforma de análisis de datos, en este caso lo emplearemos para el análisis de Software, aunque el potencial de esta herramienta abarca cualquier área de la ingeniería u otra disciplina que requiera hacer un análisis de información, en particular utilizaremos *Mondrian*[4], esta es una herramienta dentro de Moose que nos permite generar una representación gráfica de los datos, una TRAZA del sistema mediante el registro de las invocaciones de cada uno de los métodos durante el uso normal del sistema, con el fin de poder tener nuestro Profiler Multidimensional.



General workflow of working with Moose.

[3] http://www.moosetechnology.org/
[4] http://www.squeaksource.com/Mondrian/

Dada la TRAZA obtenida de la ejecución de la prueba previamente definida y guardada, podremos realizar una serie de análisis de manera más intuitiva buscando comportamientos representados gráficamente, como ejemplo, con el siguiente extracto de código, podemos importar el XML en este caso un pequeño extracto de la TRAZA, y visualizarlo, según el modelo definido.

```
Interaccion
    "    self new Interaccion    "

    | xml document view f rows rows2 |
    xml := '     <trace_Presentacion>
                    <Metodo>~/Ingreso/editarEmpresa.aspx</Metodo>
                    <Parametros></Parametros>
                    <StartTime>22:23:48</StartTime>
                        <trace_Negocio>
                            <Metodo>BOEmpresa BuscarEmpresasPorRut</Metodo>
                            <Parametros>, rut = 992342340-5</Parametros>
                            <StartTime>22:23:48</StartTime>
                            <tiempo_ms>50</tiempo_ms>
                        </trace_Negocio>
                        <trace_Negocio>
                            <Metodo>BOEmpresa BuscarEmpresasPorRut</Metodo>
                            <Parametros>, rut = 934223230-5</Parametros>
                            <StartTime>22:23:48</StartTime>
                            <tiempo_ms>50</tiempo_ms>
                        </trace_Negocio>
                    <tiempo_ms>681</tiempo_ms>
                </trace_Presentacion>  '.
    document := XMLDOMParser parseDocumentFrom: xml readStream.
    rows := document // 'trace_Presentacion'.
    view := MOViewRenderer new.
    rows
        do: [ :each |
            "Nodo trace_Presentacion "
            view shape rectangle.
            "PopUp de información "
            view interaction popupText: [:aaa | each textData , String cr,'Tiempo Promedio = ', each promedio_ms asInteger asString  ,
' ms', String cr,'Total Tiempo = ', each tiempo_ms  asInteger asString, ' ms', String cr, 'Cantidad = ', each cantidad asInteger asString ].
            view node: each.
            rows2 := each // 'trace_Negocio'.
            rows2
                do: [ :each2 |
                    "Nodo trace_Negocio "
                    view shape rectangle.
                    view node: each2.
                    view edge: each2 from: each to: each2 ] ].
    view treeLayout.
    view open
```

Código 10 - Ejemplo de Importacion TRAZA

En el resultado de la ejecución del código anterior se puede apreciar los nodos y sus relaciones de jerarquía, además al seleccionar el nodo, le hemos agregado un popup nos entrega información adicional. En el siguiente orden:

```
~/Ingreso/editarEmpresa.aspx
Tiempo Promedio = 23 ms
Total Tiempo = 138 ms
Cantidad = 6
```

- *Método*
- *Tiempo Promedio en milisegundos*
- *Total Tiempo de ejecución.*
- *Cantidad de invocaciones.*

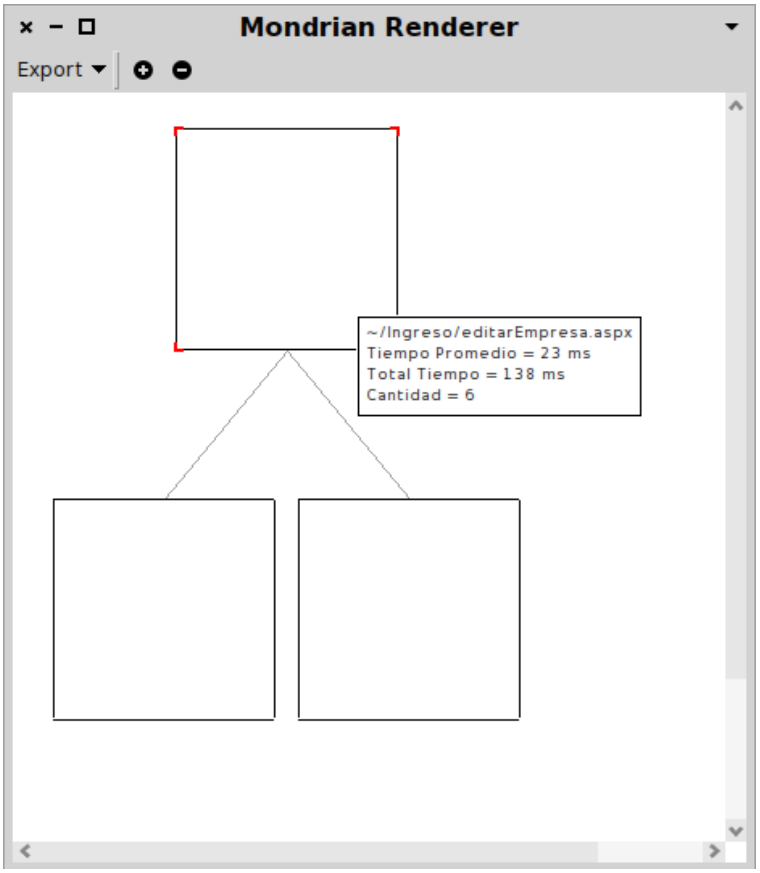


Figura 17 - Ejemplo de visualización de TRAZA

3.3 EXPERIMENTO: OPTIMIZACIÓN DE LA APLICACIÓN

En este momento tenemos una versión 1.0 del software funcionado, y se han presentamos reclamos con respecto a los tiempos de respuesta, es por esto que queremos hacer un análisis en el Profiler Multidimensional para optimizar nuestro software, y adicionalmente realizar la verificación de nuestro modelo o hipótesis. Lo primero es necesario saber cuál es la condición actual del sistema, que nos sirva de línea, utilizando la versión 1.0 del software con la actual carga del sistema, 24 meses de uso. Utilizaremos una TRAZA obtenida con la ejecución del set de pruebas que preparamos con nuestra herramienta para testing automatizado, el resultado obtenido es el siguiente. La visualización contiene un nodo principal que nos indica la siguiente información:

Versión Software	**Nombre TRAZA**	**Tiempo Promedio**	**Tiempo Total**	**Cantidad de Request**	**Cantidad de Consultas a la BD**
Versión 1.0	Log 24 Meses	2299 ms	441580 ms	192	215

Tabla 8 - Detalle resumen ejecución v1.0

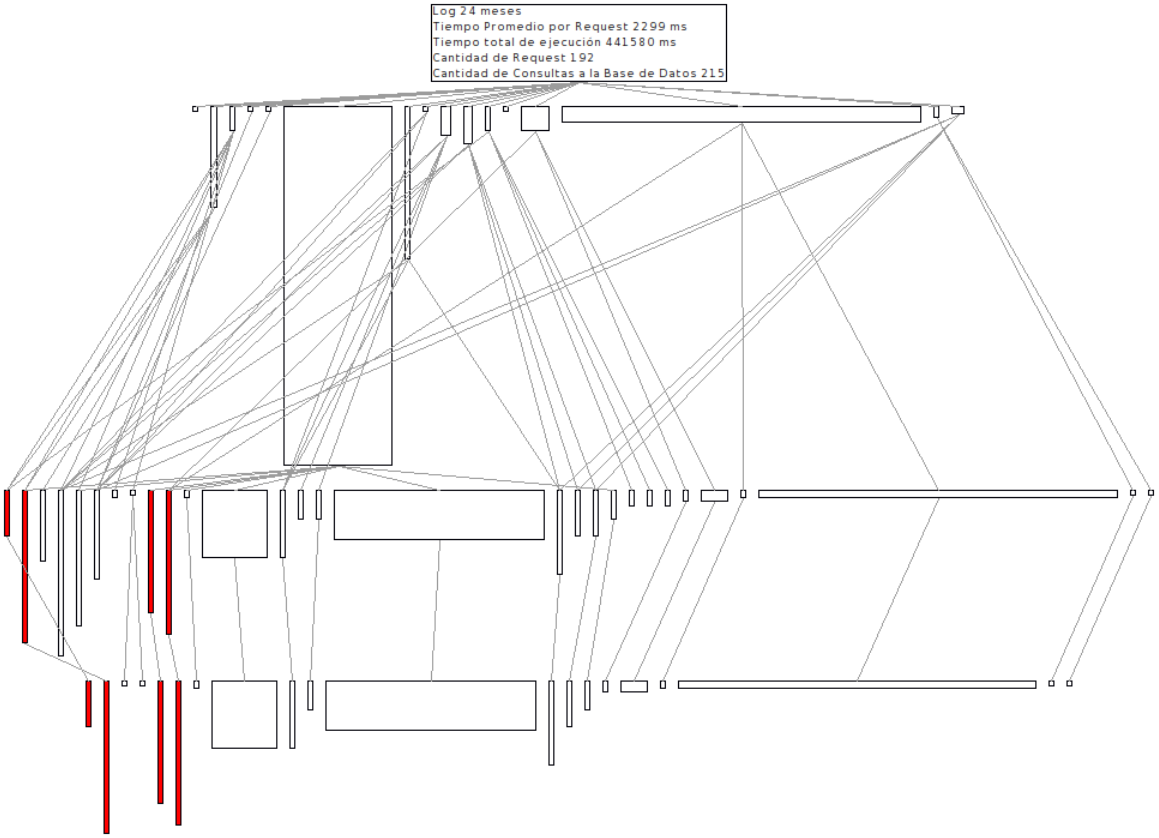


Figura 18 - Visualización - Situación Actual (Versión 1.0)

3.3.1 OPTIMIZACIONES T-SQL

A continuación utilizaremos la “Tabla 9 - Criterios de selección Nodo Capa de dato”, definida en la etapa de modelamiento, para identificar en nuestro Profiler Multidimensional, los nodos candidatos a ser optimizados.

Dimensión	**Criterio**
Arquitectura	Capa Dato
Tiempos de Respuesta	Mayores tiempos promedio en capa Datos (Ancho)
Uso	Mayor al promedio (Alto)

Tabla 10 - Criterios de selección Nodo Capa de dato

Versión	**Nombre TRAZA**	**Tiempo Promedio**	**Tiempo Total**	**Cantidad de Request**	**Cantidad de Consultas a la BD**
Versión 1.0	Escenario Actual Log 24 Meses	2299 ms	441580 ms	192	215

Tabla 11 - Detalle resumen ejecución v1.0

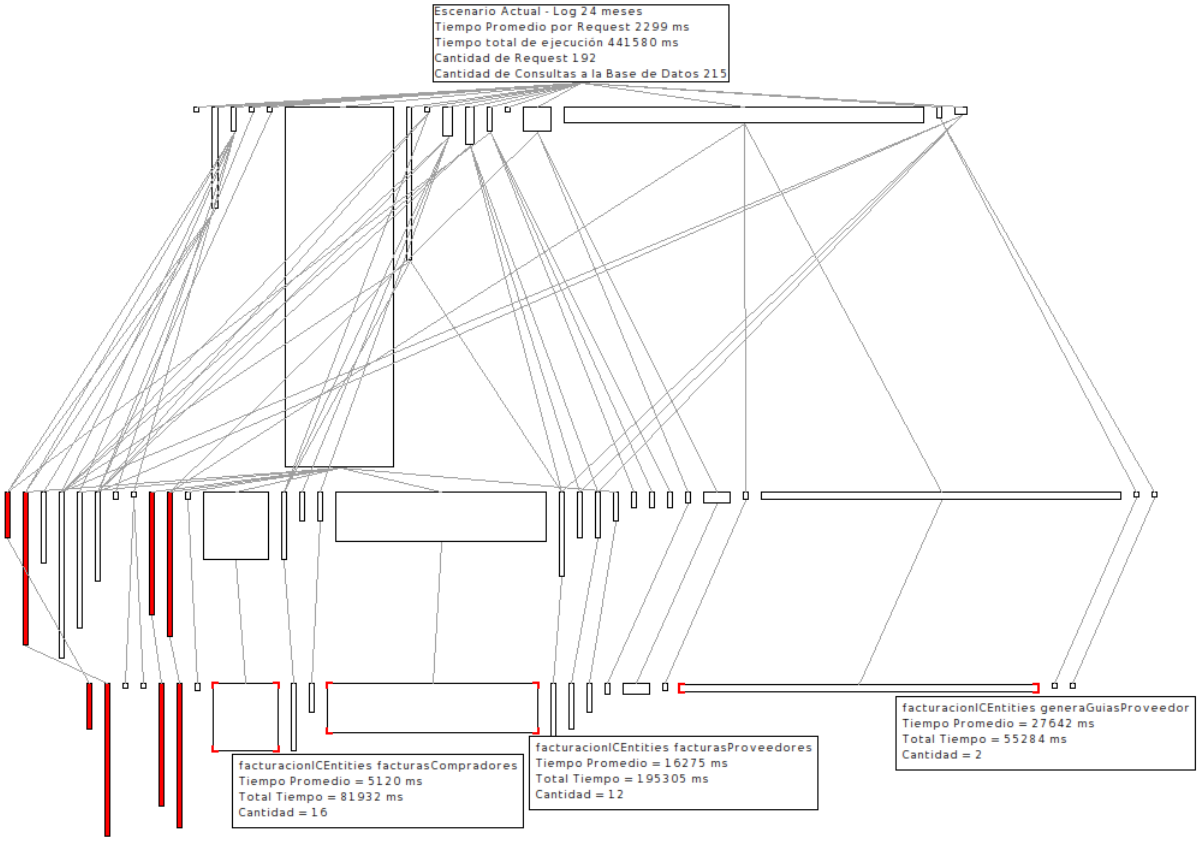


Figura 19 Visualización - Análisis Situación Actual Versión 1.0

En la siguiente imagen se ven los 3 nodos que seleccionaremos para optimizar, ya que son los 3 más anchos, 2 tienen un alto significativo que nos indica que tiene un alto uso y existe uno que a pesar de no ser alto es el más ancho, por lo tanto también lo seleccionaremos.

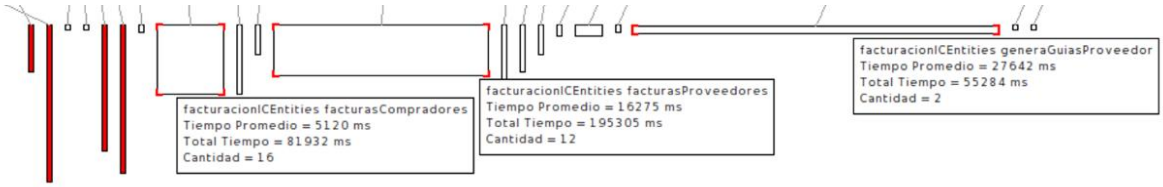


Figura 20 - Visualización, Selección Nodos Capa Dato

Método (Clase.Método)	Tiempo Promedio	Total Tiempo	Cantidad
facturacionICEntities.facturasCompradores	5120 ms	81932 ms	16
facturacionICEntities.facturasProveedores	16275 ms	195305 ms	12
facturacionICEntities.generaGuiasProveedor	27642 ms	55284 ms	2

Tabla 12 - Detalles de Nodos seleccionados en la capa datos.

De la imagen y tabla anterior los 3 nodos en la capa de datos que presentan un gran tiempo promedio de respuesta, lo cual se aprecia con un ancho proporcionalmente mayor que el resto de los nodos. Estos incluso se pueden asociar a los altos tiempos que también tienen las capas de presentación que invocan dichos métodos. Para optimizar en esta capa tenemos varias alternativas, dependiendo claro de la aplicación, y su arquitectura, en nuestro caso podría ser alguna de estas:

- Mejoras en la componente de acceso a datos.
- Creación de índices ad-hoc, para mejorar la consulta a los datos.
- Optimización de código T-SQL del procedimiento almacenado.
- Modificación de estructuras de datos(des normalización de tablas)

Nosotros en nuestro caso utilizaremos la creación de índices ad-hoc que nos permita mejorar los tiempos de respuestas para acceder a los datos, a continuación está el código donde se crea índice ad-hoc para optimizar la búsqueda de las guías de despacho de proveedores.

```
CREATE CLUSTERED INDEX [_index_GuiasDespachoProveedor] ON
[dbo].[GuiasDespachoProveedor]
(
        [idfactura] ASC,
        [Código de Producto] ASC,
        [Periodo] ASC
)WITH (SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF,
DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]
go
```

Tabla 13 -Creación de Índice Ad-Hoc. (T-SQL)

Optimizando solo los 3 nodos escogidos, logramos bajar considerablemente los tiempos de respuesta de la aplicación, se genera la Versión 1.1 del software la cual incorpora los cambios antes mencionados.

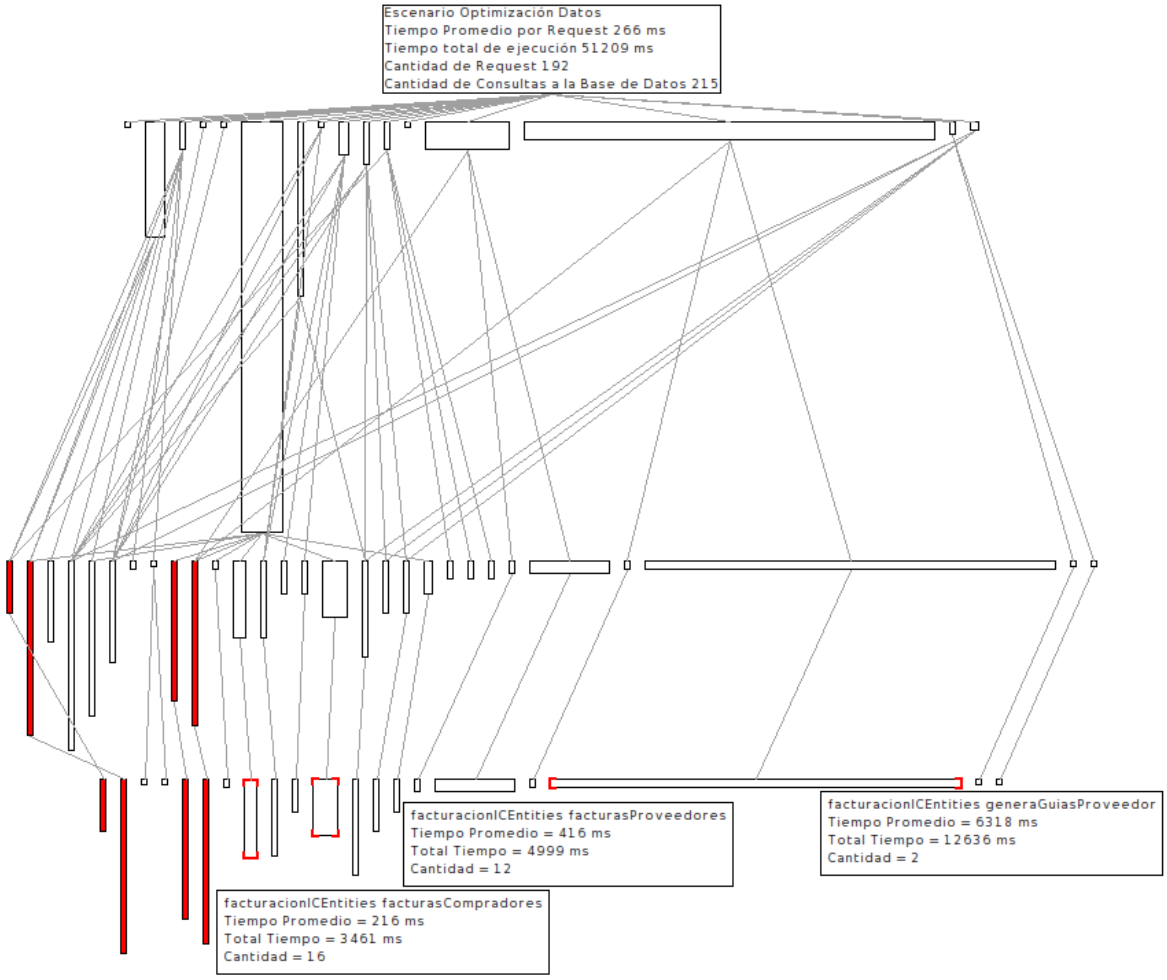


Figura 21 - Visualización, Optimización Capa Dato, Versión 1.1

Versión	Nombre TRAZA	Tiempo Promedio	Tiempo Total	Cantidad de Request	Cantidad de Consultas a la BD
Versión 1.0	Escenario Actual Log 24 Meses	2299 ms	441580 ms	192	215
Versión 1.1	Optimización Datos	266 ms	50206 ms	192	215

Tabla 14 - Detalle resumen ejecuciones; v1.0, v1.1

Método (Clase.Método)	Tiempo Promedio	Total Tiempo	Cantidad
facturacionICEntities.facturasCompradores	216 ms	3461 ms	16
facturacionICEntities.facturasProveedores	416 ms	4999 ms	12
facturacionICEntities.generaGuiasProveedor	6318 ms	12636 ms	2

Tabla 15 - Detalles de Nodos optimizados en la capa datos.

3.3.2 MANEJO DE CACHÉ

Una vez realizados los cambios en la Capa de Datos, podemos enfocar nuestros esfuerzos en la capa de Negocios, en este caso se distinguen los nodos en rojo, estos indican que tienen el 100% de las invocaciones con los mismos parámetros y adicionalmente más de una invocación, por lo tanto se presume que es posible mantener los datos en Cache, con el fin de minimizar los accesos a la base de datos, reduciendo así los tiempos de respuestas, y por consiguiente, aumentar la concurrencia y disponibilidad de la aplicación.

Dimensión	**Criterio**
Arquitectura	Capa Negocio
Variabilidad	Dato Invariable (Color Rojo)
Uso	Mayor uso en la capa. (Alto)

Tabla 16 - Criterios de selección Nodo Capa de dato

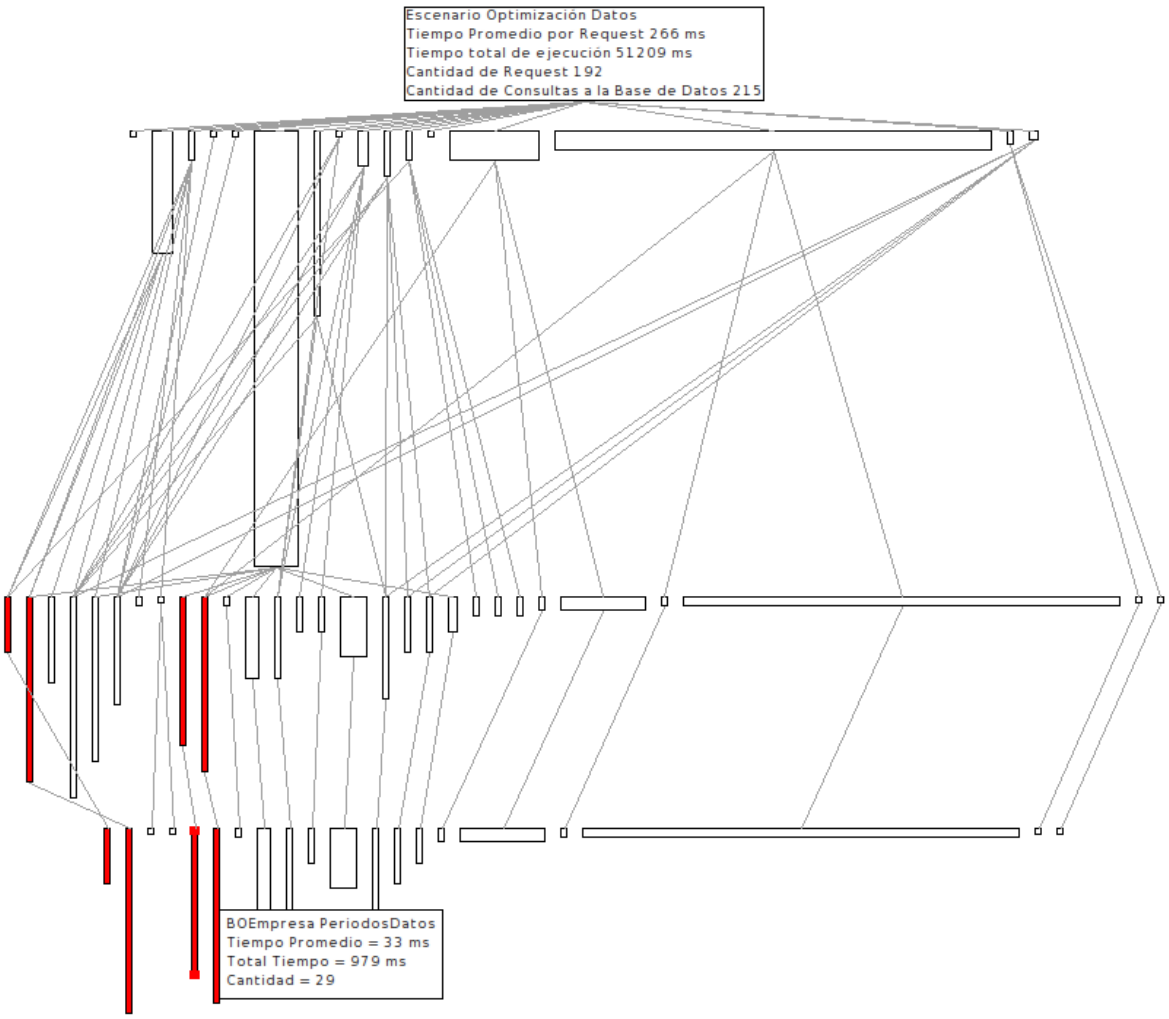


Figura 22 - Visualización análisis versión 1.1, Manejo de Caché

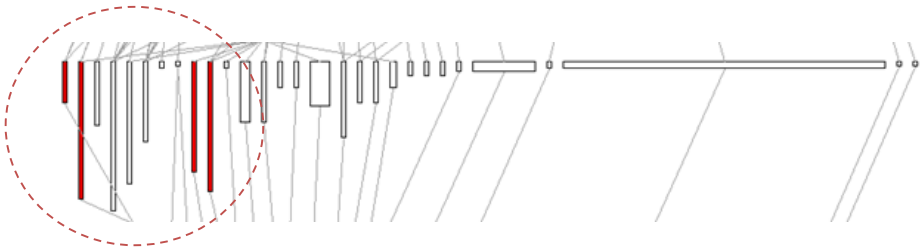


Figura 23 - Visualización Capa de Negocio, Versión 1.1, Manejo de Caché

De la imagen anterior se aprecian 4 Nodos en la capa de Negocios que presentan un gran número de invocaciones lo cual se puede ver por el alto sobre el promedio y además estos son invocados en un 100% de las veces con los mismos parámetros y tiene más de 1 invocación, lo cual se aprecia con el color rojo de los nodos. Adicionalmente, aunque no es este el caso, podría en esta capa existir elementos que tengan altos tiempos de respuestas, los cuales deberían también considerarse para la optimización. Para optimizar en esta capa utilizaremos las siguientes optimizaciones:

- Utilización de Caché, para disminuir el número de Consultas a la Capa de datos.
- Optimización de código C#.

Para nuestro sistema la solución fue la utilización de caché, al cual le daremos un tiempo fijo de expiración, en este ejemplo es del 5 minutos, pero es configurable dependiendo del software a analizar.

En el primer método, se puede ver cómo se va a buscar en primera instancia al caché, si no existe, se realiza la consulta a la base de datos y se guarda en caché.

```
public static List<string> Periodos()
{
      if (HttpContext.Current.Cache["PeriodosDatos"] != null)
      {
        return (List<string>)HttpContext.Current.Cache["PeriodosDatos"];
      }
        else
      {
        var value = PeriodosDatos();
        HttpContext.Current.Cache.Insert("PeriodosDatos",
                                value, null, System.Web.Caching.Cache.NoAbsoluteExpiration,
                                TimeSpan.FromMinutes(5));
        return (List<string>)HttpContext.Current.Cache.Get("PeriodosDatos");
       }
}
```

Figura 24 - Utilización de caché. (c#)

En el siguiente método, se busca evitar los side effects, debido a la posible actualización de la data durante la validez del dato en la caché, por lo tanto ante alguna modificación que implique la data en caché, eliminamos esta de caché, para que sea nuevamente consultada a la base de datos. Cabe señalar que se deben evitar realizar cambios directamente sobre la base de datos, dado que estos no se verían reflejados hasta que el dato en caché expire, para prevenir esto se debe reciclar el servidor web, lo cual elimina la data de cache.

```
public static bool UpdatePeriodo(Periodo periodo)
{
      ICModel.facturacionICEntities context = new facturacionICEntities();
      context.actualizaPeriodo(periodo);
            if (insertaEmpresaComprador(periodo))
            {
               HttpContext.Current.Cache["PeriodosDatos"] != null;
               return true;
            }
            else
               return false;
}
```

Código 11 - Utilización de caché, evitando Side Effects. (c#)

Optimizando solo los 4 nodos escogidos, se genera la Versión 1.2 del software la cual incorpora los cambios antes mencionados.

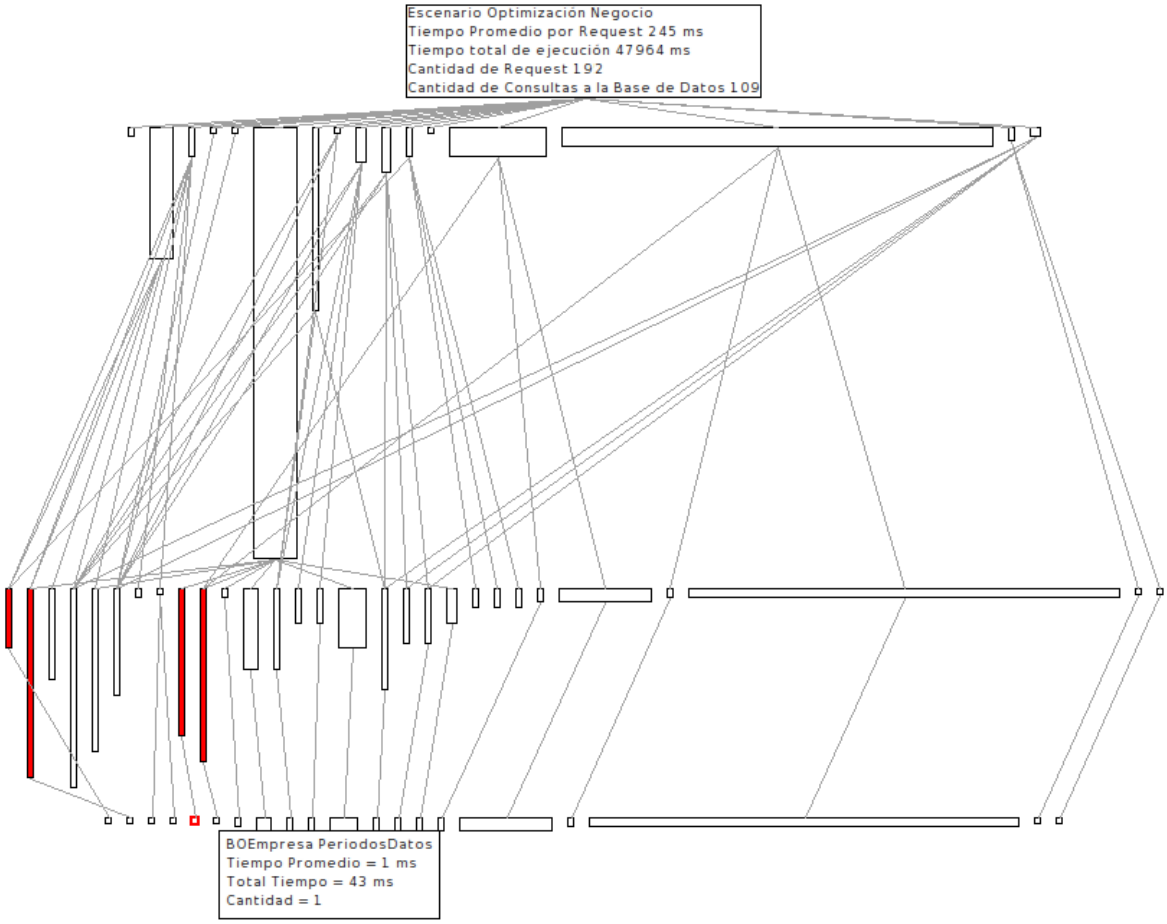


Figura 25 – Visualización - Optimización Capa Negocio (manejo de caché), Versión 1.2

Versión	Nombre TRAZA	Tiempo Promedio	Tiempo Total	Cantidad de Request	Cantidad de Consultas a la BD
Versión 1.0	Escenario Actual Log 24 Meses	2299 ms	441580 ms	192	215
Versión 1.1	Optimización Datos	266 ms	50206 ms	192	215
Versión 1.2	Optimización Negocio	245 ms	47964 ms	192	109

Tabla 17 - Detalle resumen ejecuciones; v1.0, v1.1, v1.2

Como respuesta a los cambios realizados, se puede ver una pequeña disminución en los tiempos de respuesta, y adicionalmente el número de consultas a la base de datos disminuyó casi a la mitad. Esto se debe a que el dato que originalmente se iba a buscar a la base de datos ahora se mantiene en la sesión, mejorando así la disponibilidad de la base de datos.

En la imagen comparativa entre la versión 1.1 a la izquierda y la versión 1.2 a la derecha se puede ver que desde la capa de negocio antes se invocaba a la capa de datos cada vez que requería un dato, ahora con la nueva versión esto solo ocurre una vez, ya que posteriormente el valor es guardado en memoria cache para su rápido acceso.

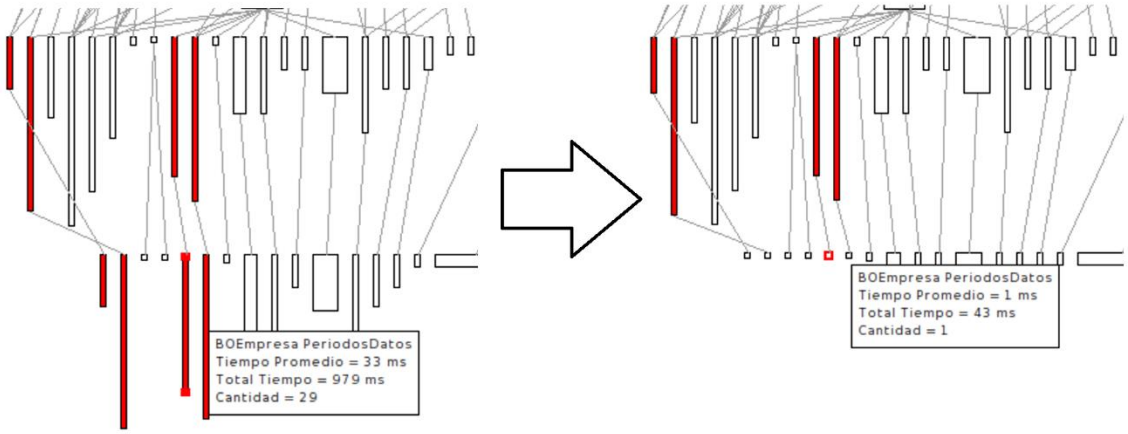


Figura 26 - Comparación Versión 1.1 (izq) versus Versión 1.2 (der)

3.3.3 MEJORA INTERFAZ DE USUARIO

Una vez realizados los cambios en la Capa de Negocios, podemos enfocar nuestros esfuerzos en la capa de Presentación, en este caso, dado que la capa de presentación por lo general no procesa mayor información, lo que debemos buscar la disminución de la cantidad de request innecesarios, en este caso enfocaremos nuestros esfuerzos en el nodo "~/ingreso/buscarFacturas.aspx".

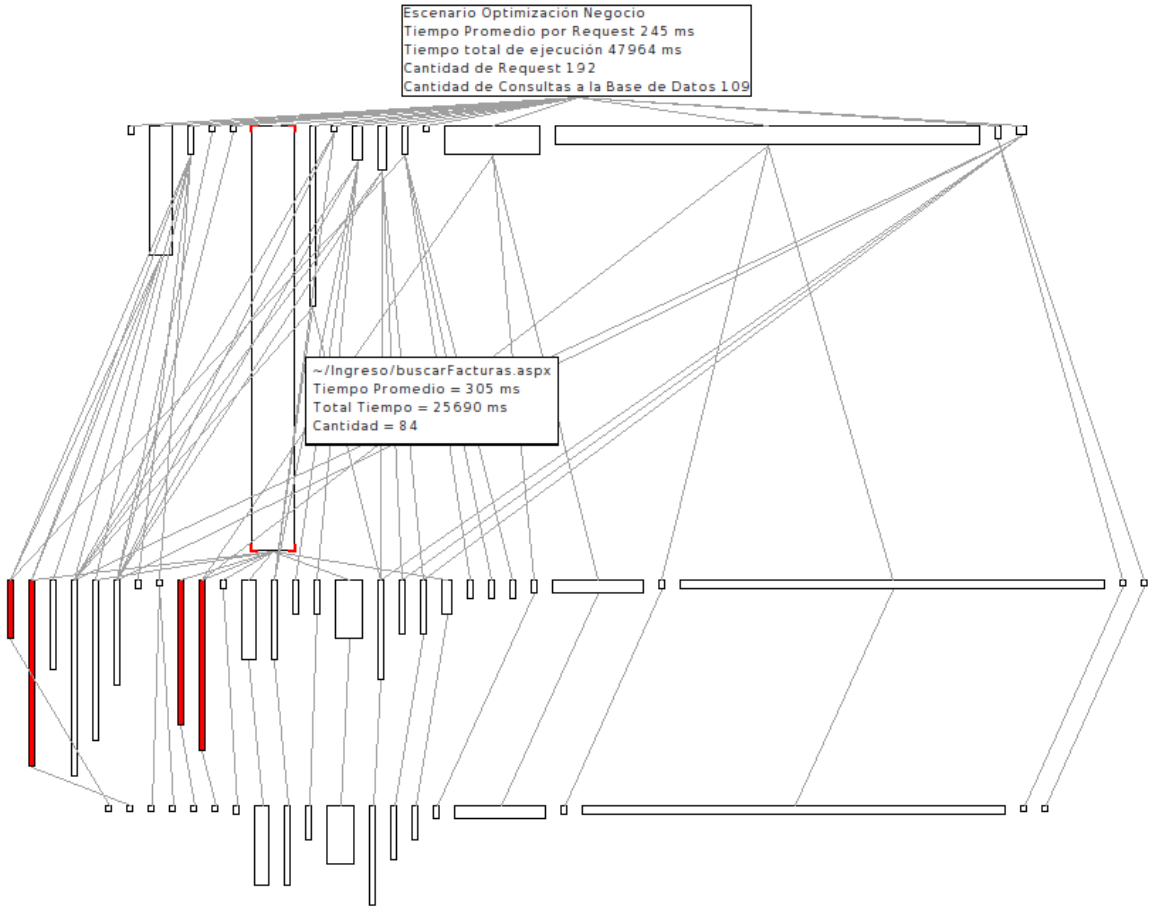


Figura 27- Visualización Análisis Versión 1.2, Mejora Interfaz de usuario

Método	Tiempo Promedio	Total Tiempo	Cantidad
~/Ingreso/buscarFacturas.aspx	305 ms	25690 ms	84

Tabla 18 - Detalles de Nodo a optimizar en la capa Presentación.

De la imagen anterior se aprecia un Nodo en la capa de Presentación que destaca por un gran número de invocaciones lo cual se aprecia dado el alto de la figura. Para optimizar en esta capa utilizaremos las siguientes optimizaciones:

- Disminución de utilización de Postback.
- Utilización de JavaScript/html5 para hacer validaciones del lado del cliente.

Para nuestra página, utilizaremos esta función nos permite mediante JavaScript identificar a través de la propiedad event.keyCode la tecla digitada. Si el carácter digitado no se encuentra dentro del rango de teclas numéricas (del 48 al 57) retorna falso (no permite escribir el carácter), como lo muestra el siguiente código:

```
function ValidaSoloNumeros()
 {
if ((event.keyCode < 48) || (event.keyCode > 57))
event.returnValue = false;
}
```

Código 12 - Validación en browser solo números. (JavaScript)

Llamamos la función en el evento onkeypress de la caja de texto:

```
<input id="txNumero" name="txNumero" onkeypress="ValidaSoloNumeros()" type="text">
```

Código 13 - Validación en cliente (HTML)

Optimizando el nodo escogido, se genera la Versión 1.3 del software la cual incorpora los cambios antes mencionados.

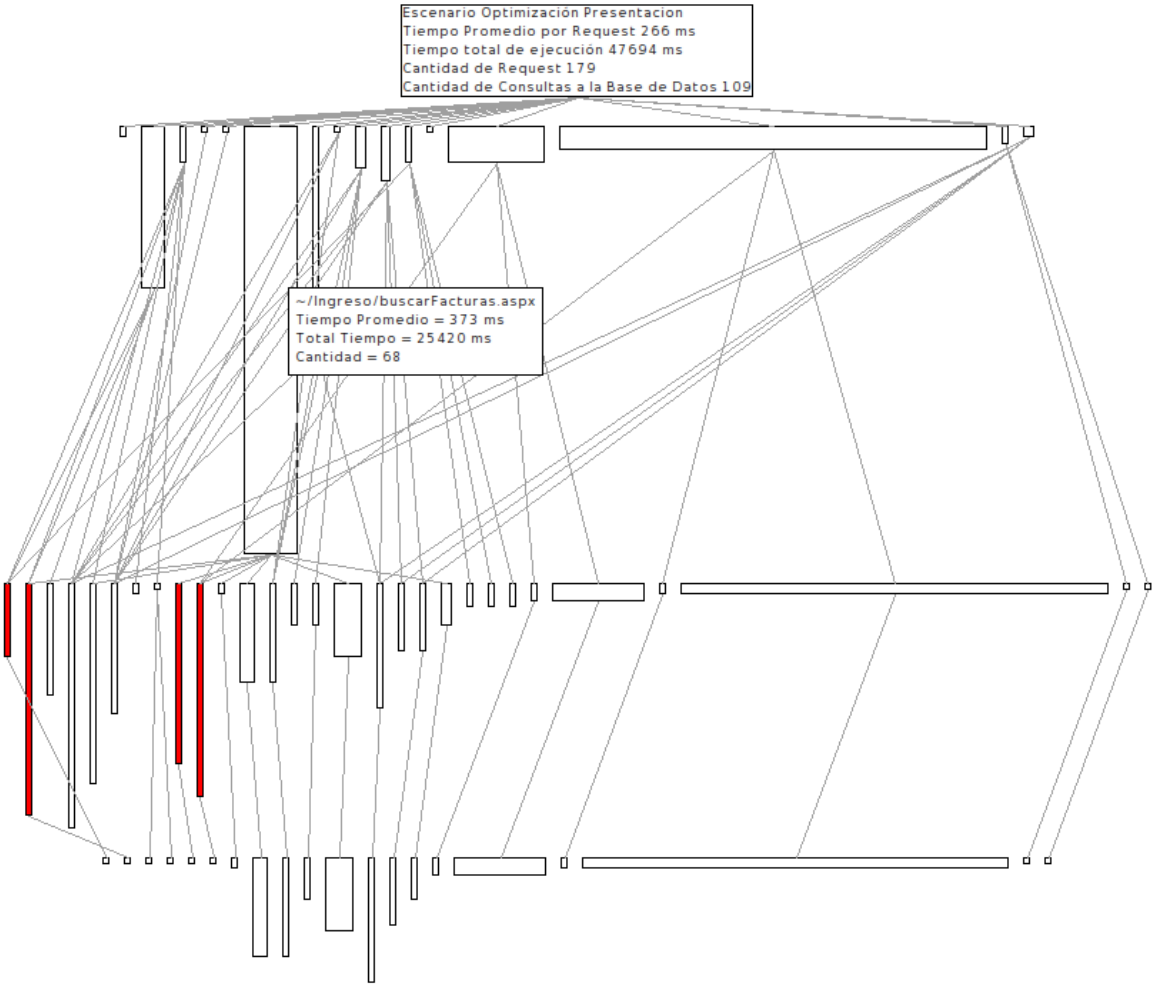


Figura 28 – Visualización - Optimización Capa Presentación, Versión 1.3

Versión	Nombre TRAZA	Tiempo Promedio	Tiempo Total	Cantidad de Request	Cantidad de Consultas a la BD
Versión 1.0	Escenario Actual Log 24 Meses	2299 ms	441580 ms	192	215
Versión 1.1	Optimización Datos	266 ms	50206 ms	192	215
Versión 1.2	Optimización Negocio	245 ms	47964 ms	195	109
Versión 1.3	Optimización Presentación	266 ms	47694 ms	179	109

Tabla 19 - Detalle resumen ejecuciones; v1.0, v1.1, v1.2, v1.3

Método	Tiempo Promedio	Total Tiempo	Cantidad
~/Ingreso/buscarFacturas.aspx	373 ms	25420 ms	68

Tabla 20 - Detalles de Nodo optimizado en la capa Presentación.

Con respecto a los tiempos de respuesta promedio, a pesar que parece que aumentaron los tiempos promedios, el tiempo total de ejecución de la prueba disminuyó cerca de 300ms, adicionalmente la disminución de Request, nos permite descongestionar el servidor web, aumentando la concurrencia y disponibilidad del sistema en general.

3.3.4 RESULTADOS

Durante el desarrollo de las optimizaciones podemos ver como en cada iteración se reflejan distintos comportamientos con respecto a la capa en la cual se realiza la optimización, en el caso de la capa de datos, nos permite disminuir los tiempos de respuesta de las consultas a la base de datos, por otro lado en la capa de negocio, mediante la utilización de caché, podemos ver como disminuyen los accesos a la base de datos, y por último en la capa de presentación nos permite disminuir los Request.

Versión	**Nombre TRAZA**	**Tiempo Promedio**	**Tiempo Total**	**Cantidad de Request**	**Cantidad de Consultas a la BD**
Versión 1.0	Escenario Actual Log 24 Meses	2299 ms	441580 ms	192	215
Versión 1.1	Optimización Datos	266 ms	50206 ms	192	215
Versión 1.2	Optimización Negocio	245 ms	47964 ms	192	109
Versión 1.3	Optimización Presentación	266 ms	47694 ms	179	109

Tabla 21 - Detalle resumen ejecuciones; v1.0, v1.1, v1.2, v1.3

En el siguiente gráfico están representado los tiempos promedio de ejecución del sistema, en las distintas versiones de este. Y se puede ver la diminución considerable de los tiempos de respuesta en la versión 1.1, La cual se encargó de optimizar el acceso a los datos, utilizando índices ad-hoc para las consultas con problemas.

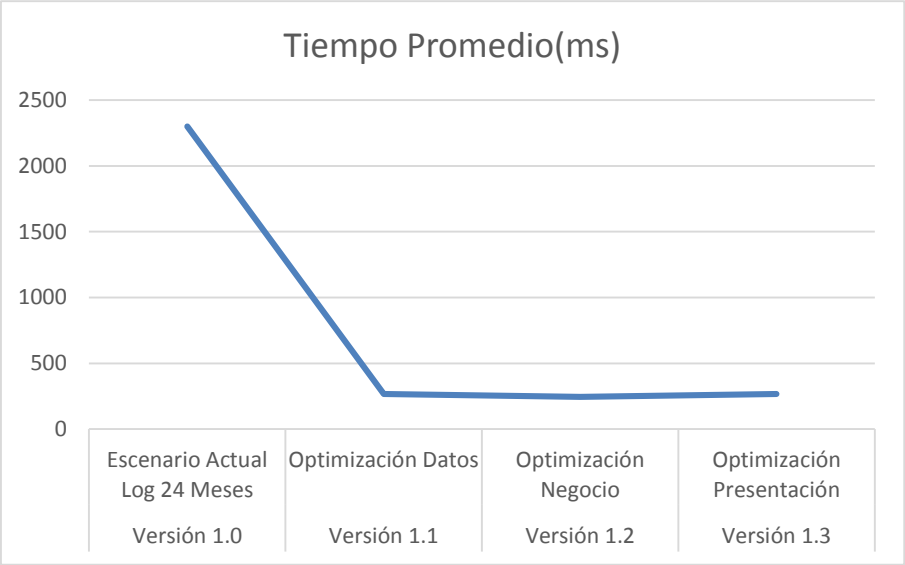


Gráfico 2- Comparación tiempos promedio, de todas las versiones

En el siguiente gráfico al igual que el grafico anterior que están representado los tiempos del sistema, en las distintas versiones de este. Y se puede ver la diminución considerable de los tiempos de respuesta en la versión 1.1, La cual se encargó de optimizar el acceso a los datos, utilizando índices ad-hoc para las consultas con problemas. Este comportamiento se debe a que ambas métricas están asociadas a la misma dimensión.

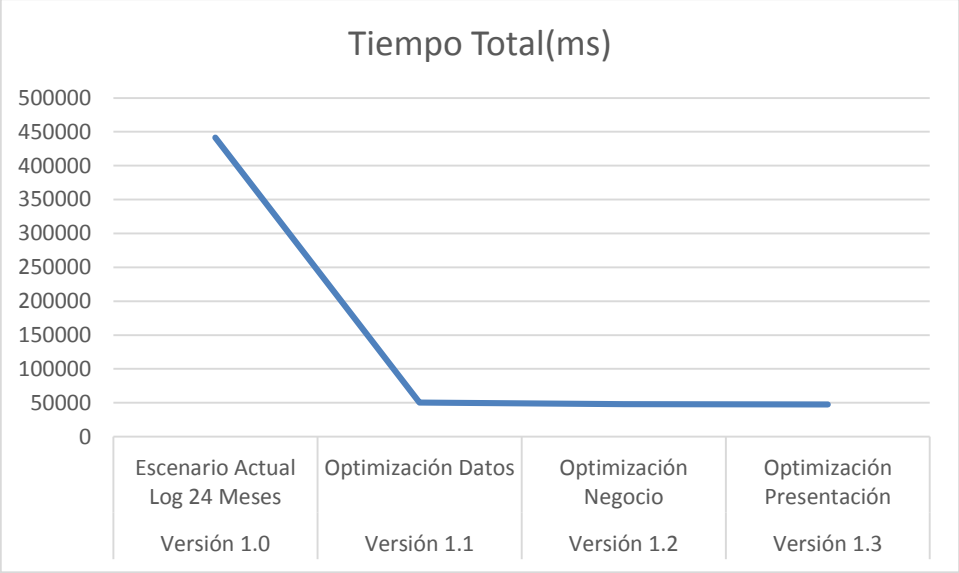


Gráfico 3 - Comparación tiempos totales, de todas las versiones

Si revisamos el gráfico número 3, donde el punto evaluado son la cantidad de consultas a la BD, podemos ver que al momento de aplicar la lógica de optimización sobre los métodos de la capa de negocio, pudimos disminuir la carga de nuestro servidor de base de datos.

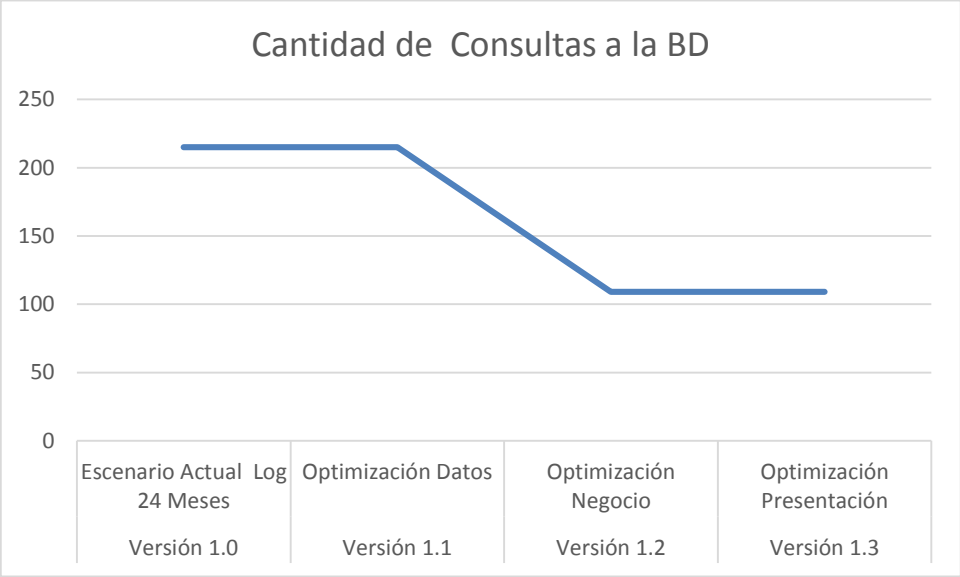


Gráfico 4 - Comparación Cantidad de Consultas a la BD, de todas las versiones

Los request son las invocaciones que nos hace a nuestra capa de presentación, mediante el uso del browser, cada vez que un cliente, en este caso un navegador web, envía un request nuestro servidor web debe atender dicho llamado y destinarle recursos. Con la optimización que hicimos en una página, utilizando mejor las validaciones del lado del cliente, liberamos recursos a nuestro servidor web, permitiéndonos así aumentar la disponibilidad del servicio.

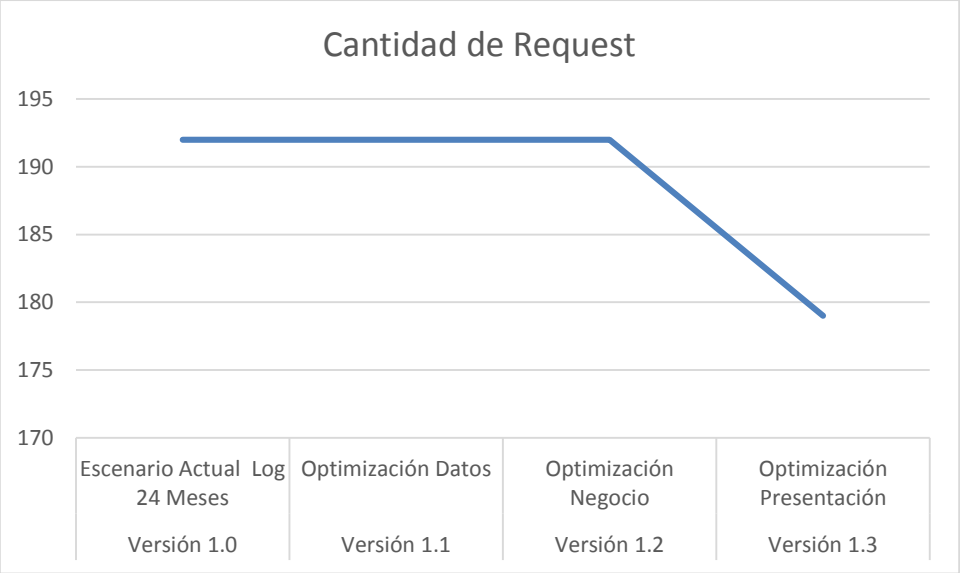


Gráfico 5 - Comparación Cantidad de Request, de todas las versiones

3.4 EXPERIMENTO: COMPORTAMIENTO DEL SOFTWARE EN EL TIEMPO

Hemos dicho que el problema de rendimiento o tiempos de respuesta de nuestro aplicativo, está dado por dado por el aumento de la cantidad de datos, es por esto que en el siguiente experimento, compararemos la misma versión de software con distinto volumen de carga, es por esto que utilizaremos la versión 1.0 del software con un respaldo de la base de datos con tan solo 5 meses de carga como punto inicial de comparación, con el fin de contrarrestarlo con el actual nivel de carga del sistema (24 meses), al contar con información cuantitativa con respecto al funcionamiento del sistema, podremos desambiguar lo relatado por los usuarios con respecto a la "lentitud" del sistema y llevarlo a una métrica que nos permita un mejor control.

Versión Software	Nombre TRAZA	Tiempo Promedio	Tiempo Total	Cantidad de Request	Cantidad de Consultas a la BD
Versión 1.0	Log 5 Meses	738 ms	143991 ms	192	215

Tabla 22 - Detalle resumen ejecuciones; v1.0 con 5 meses de carga

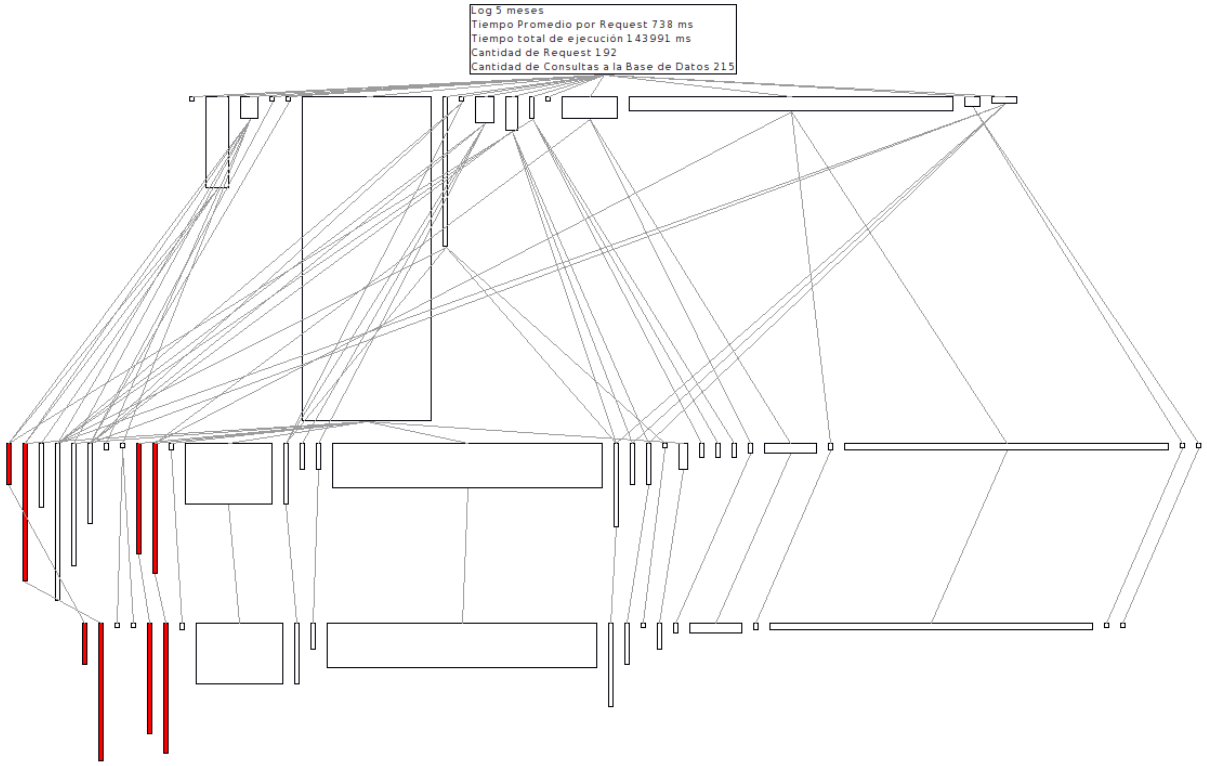


Figura 29– Visualización - Versión 1.0 con 5 meses de carga

A continuación podemos ver como aumentaron los tiempos de respuesta en nuestra versión actual del sistema, la misma versión 1.0 pero ahora con 24 meses de carga.

Versión Software	Nombre TRAZA	Tiempo Promedio	Tiempo Total	Cantidad de Request	Cantidad de Consultas a la BD
Versión 1.0	Log 24 Meses	2299 ms	441580 ms	192	215
Versión 1.0	Log 5 Meses	738 ms	143991 ms	192	215

Tabla 23 - Detalle resumen ejecución v1.0 con 5 meses de carga versus 24 Meses de carga

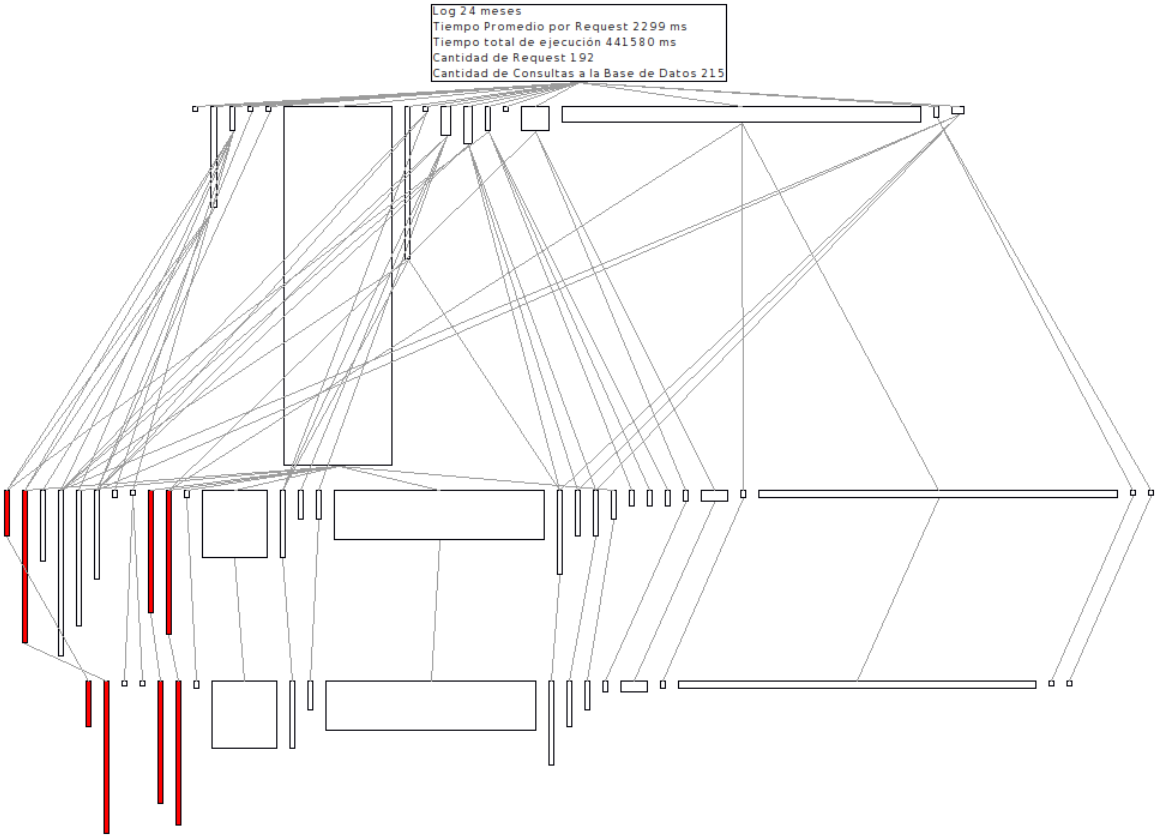


Figura 30 - Visualización - Versión 1.0 con 5 meses de carga (Situación actual)

3.4.1 RESULTADOS

Como se puede ver en la Tabla 24, los tiempos de respuesta se incrementaron considerablemente, dado que la versión del software es la misma, se puede deducir como dijimos anteriormente que el software en su diseño original no consideraba el aumento de data como un punto relevante de performance. Eso no implica que fuera un mal diseño ni que no se hiciera el control de calidad necesario, ya que los tiempos de respuesta del software en sus primeros meses de uso eran aceptables.

Versión Software	Nombre TRAZA	Tiempo Promedio	Tiempo Total	Cantidad de Request	Cantidad de Consultas a la BD
Versión 1.0	Log 24 Meses	2299 ms	441580 ms	192	215
Versión 1.0	Log 5 Meses	738 ms	143991 ms	192	215

Tabla 24 - Detalle resumen ejecución v1.0 con 5 meses de carga versus 24 Meses de carga

Como se puede apreciar en el gráfico siguiente el aumento de casi 3 veces en los tiempos promedios de respuesta, al igual que el tiempo total de ejecución de la prueba, esto indica que el principal problema fue el aumento de la carga de datos.

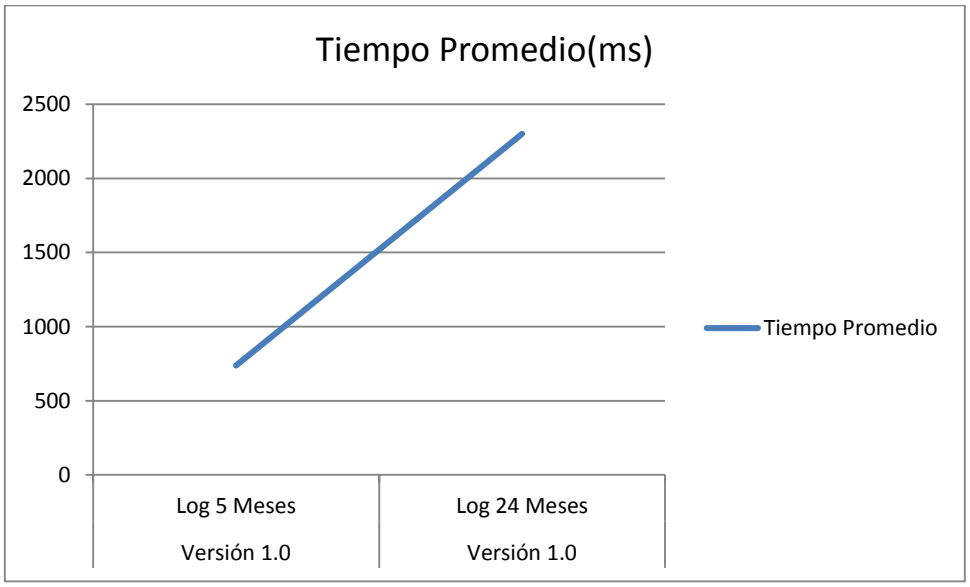


Gráfico 6 - Tiempos promedio en milisegundos, entre versión 1.0 con 5 y 24 meses de carga

3.5 EXPERIMENTO: COMPORTAMIENTO DEL SOFTWARE MEJORA DE HARDWARE

Como solución a nuestra problemática con respecto a los tiempos de respuesta del sistema, se suele proponer un mejoramiento de la infraestructura (Hardware), es por esto que incluimos un tercer experimento, en el cual se consiste en potenciar la máquina virtual donde se ejecuta el software, aumentando al doble su capacidad de procesamiento y también el doble de su memoria RAM. Cabe señalar que no hubo cambios en el storage, en el cual se encuentra la data. En las siguientes figuras se puede ver un resumen de ambos escenarios.

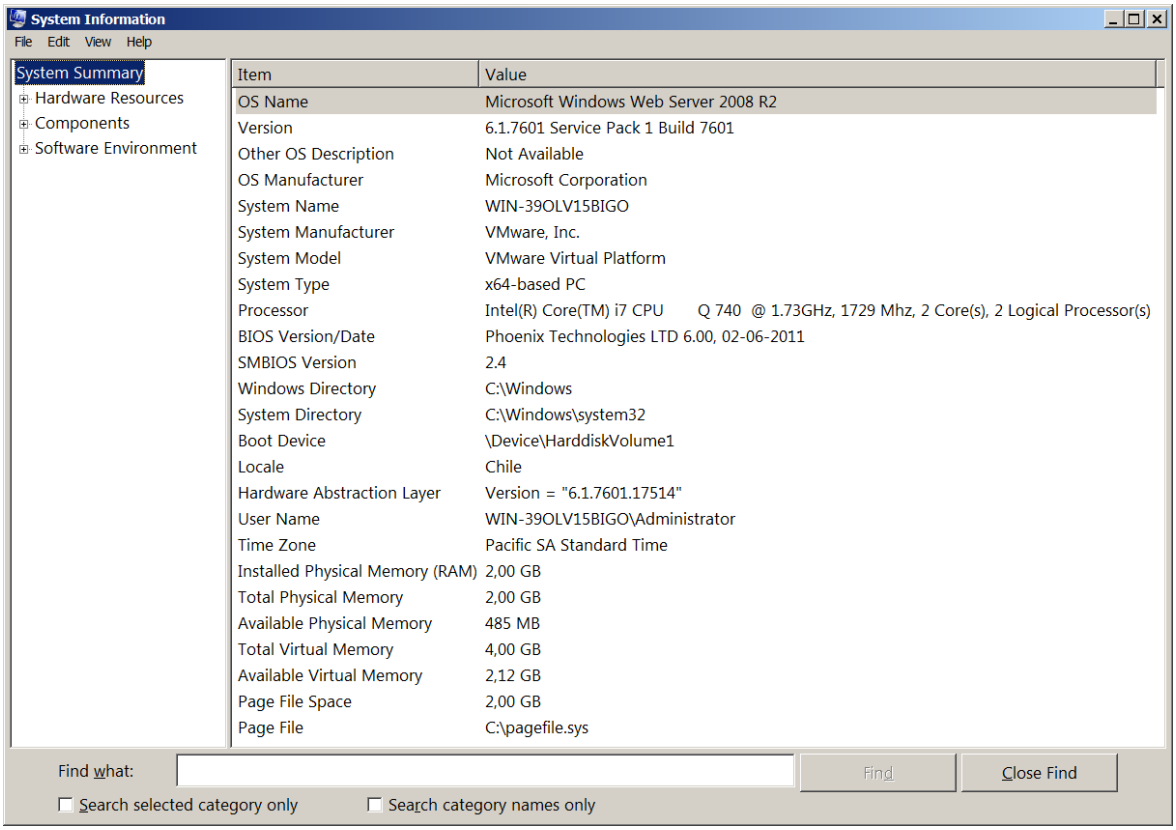


Figura 31 - Hardware Estándar de la Aplicación

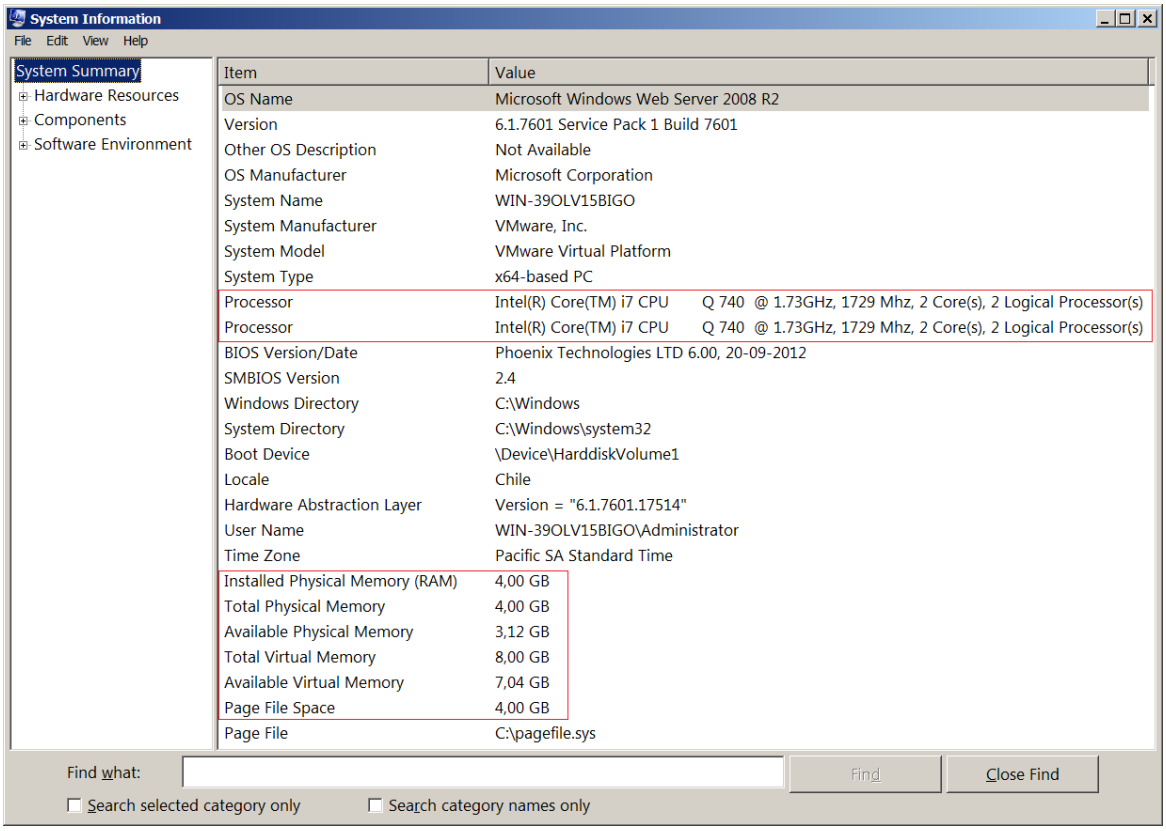


Figura 32 - Hardware Potenciado de la Aplicación

Como se aprecia en las figuras anteriores, se aumentó al doble la capacidad de procesamiento y el doble de la memoria RAM del servidor, para así en ambos escenarios ejecutar la misma prueba, con el fin de poder comparar los tiempos de ejecución de la aplicación, y ver si la posibilidad de aumentar los recursos es una solución a nuestra problemática, que en el papel resulta ser la más rápida de las solución.

A continuación ejecutamos nuestro Profiler Multidimensional, con la versión 1.0 del software y con la carga actual (24 meses). Esto sobre nuestro Hardware actual.

Versión Software	Nombre TRAZA	Tiempo Promedio	Tiempo Total	Cantidad de Request	Cantidad de Consultas a la BD
Versión 1.0	Hardware Actual Log 24 Meses	2299 ms	441580 ms	192	215

Tabla 25 - Detalle resumen ejecución v1.0 con hardware actual

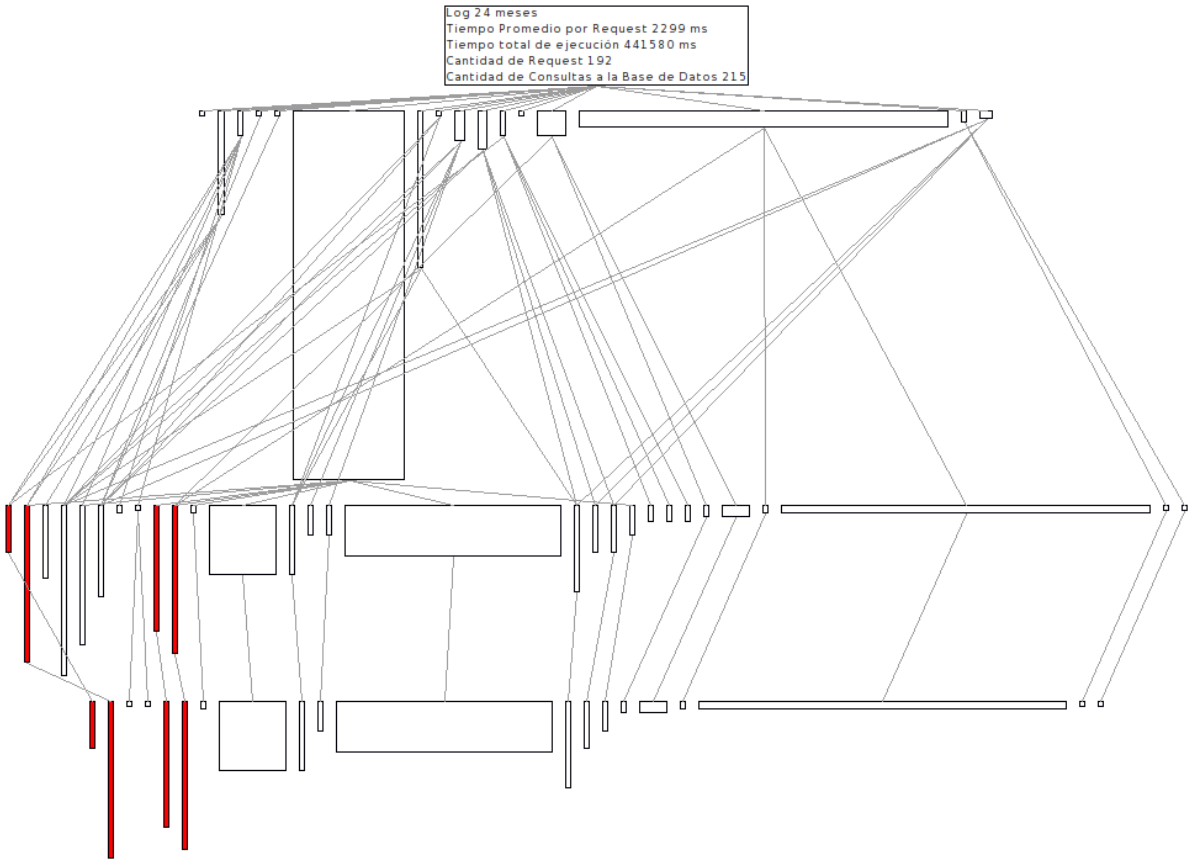


Figura 33– Visualización - Versión 1.0 con hardware actual

A continuación podemos la ejecución de nuestro Profiler Multidimensional de la misma versión 1.0 con la misma cantidad de carga (24 meses) pero en nuestro hardware potenciado.

Versión Software	Nombre TRAZA	Tiempo Promedio	Tiempo Total	Cantidad de Request	Cantidad de Consultas a la BD
Versión 1.0	Hardware Actual Log 24 Meses	2299 ms	441580 ms	192	215
Versión 1.0	Hardware Potenciado Log 24 Meses	2065 ms	396622 ms	192	215

Tabla 26 - Detalle resumen ejecución v1.0 con hardware actual ver Hardware Potenciado

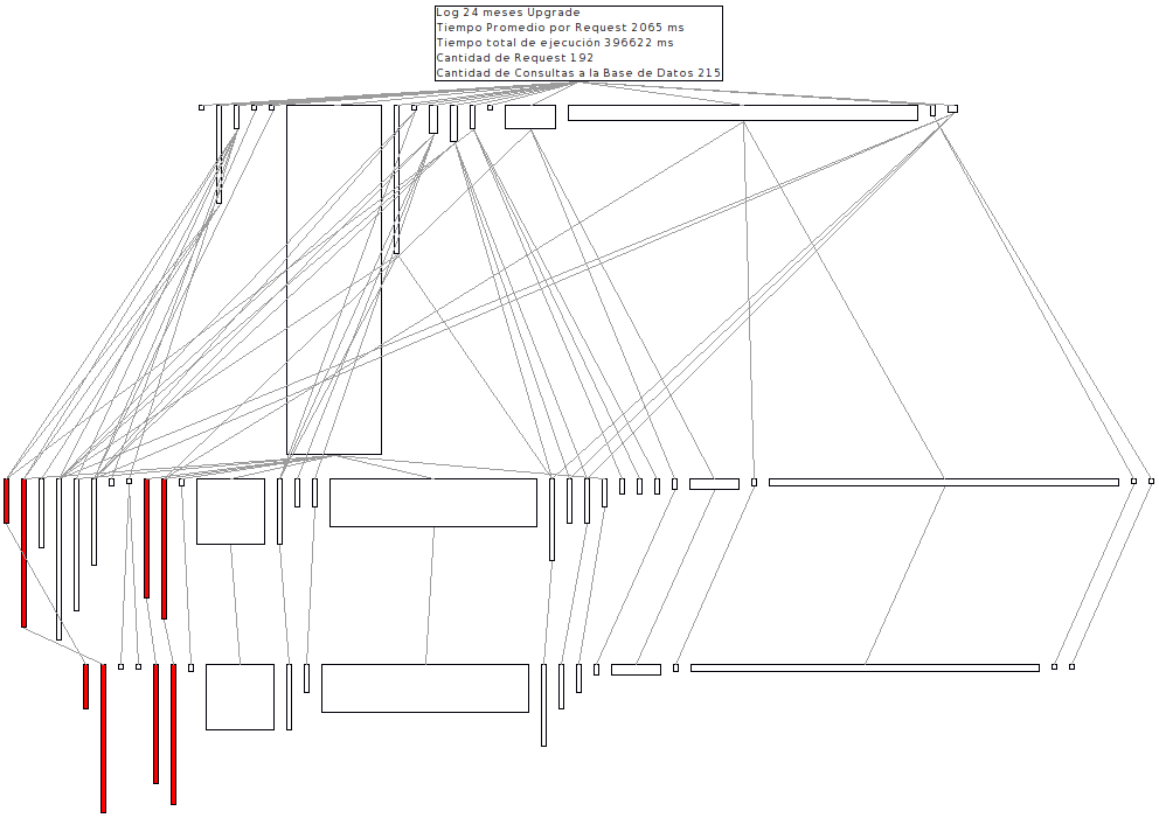


Figura 34 - Visualización - Versión 1.0 con Hardware Potenciado.

3.5.1 Resultados

Como se puede ver en la Tabla 27, si bien es cierto los tiempos de respuesta disminuyeron en cuando se ejecutó la prueba en nuestro hardware potenciado, la disminución es bastante baja, estamos hablando de cerca de un 10% de mejora, lo cual si lo comparamos con la mejora obtenida con los cambios en el código, la solución de aumento de hardware no se ve como una alternativa real.

Versión Software	Nombre TRAZA	Tiempo Promedio	Tiempo Total	Cantidad de Request	Cantidad de Consultas a la BD
Versión 1.0	Hardware Actual Log 24 Meses	2299 ms	441580 ms	192	215
Versión 1.0	Hardware Potenciado Log 24 Meses	2065 ms	396622 ms	192	215

Tabla 27 - Detalle resumen ejecución v1.0 con hardware actual ver Hardware Potenciado

Como se puede apreciar en el gráfico siguiente se puede apreciar una disminución de los tiempos promedios de respuesta, al igual que el tiempo total de ejecución de la prueba, el porcentaje de disminución alcanza al 10%. Dado que el upgrade realizado solo abarcó el aumento de la capacidad de procesamiento (CPU) y la memoria (RAM), podríamos deducir que el problema está en el storage y en el aplicativo.

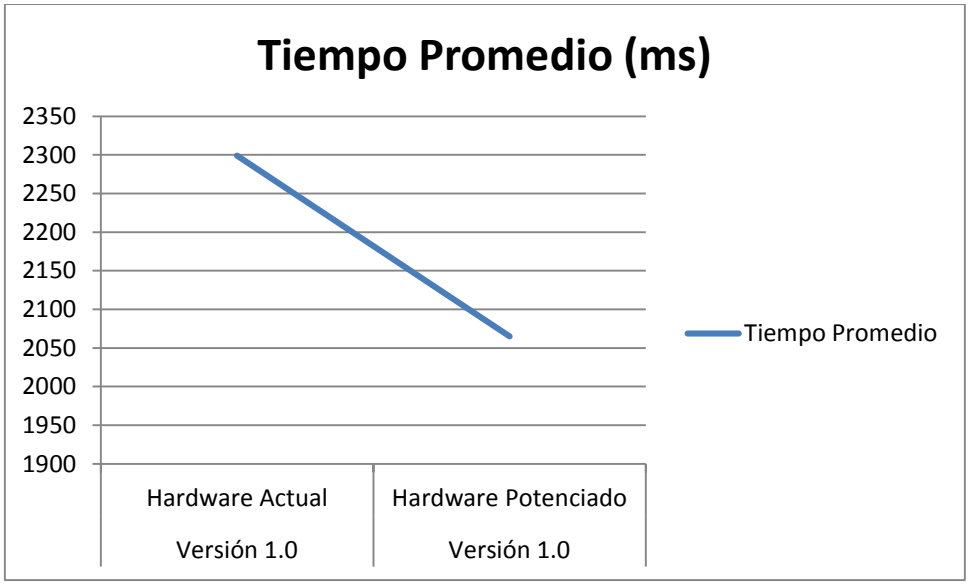


Gráfico 7 - Tiempos promedio en milisegundos, entre versión 1.0 hardware actual versus potenciado

CAPÍTULO 4: CONCLUSIONES

En este estudio se logró mejorar los tiempos de respuesta del sistema de facturación, mediante la identificación de cuellos de botella, para realizar lo anterior utilizamos el Profiler Multidimensional definido, el cual responde al modelo generado a partir de las distintas dimensiones que se querían representar sobre el comportamiento del sistema. Estas dimensiones consideradas fueron elegidas con el fin de buscar problemas conocidos, para aplicarle mejoras probadas. En un principio se planteó de manera inversa, es decir buscar comportamientos sospechosos y optimizarlos, sin embargo, al no saber con qué nos podríamos encontrar no nos aseguraba que podríamos mejorar el sistema de manera rápida, dado que no sabíamos que nos enfrentaríamos, podría llegar a ser una modificación de la arquitectura de la aplicación como solución. Es por esto, que primero se buscaron los problemas típicos y a estos aplicaríamos soluciones ya conocidas, con esto podríamos reaccionar a tiempo para generar mejoras de manera rápida y efectiva.

Los distintos problemas fueron identificados mediante el análisis de 1 o más dimensiones de manera simultánea, con esto, se pudo identificar, priorizar y definir la estrategia de optimización para el cuello de botella identificado. Todo esto fue posible gracias al conocimiento ya existente en el equipo de trabajo, sobre la arquitectura y el funcionamiento del aplicativo.

Como se puede ver en la tabla siguiente la disminución principal en los tiempos de respuesta fue en la versión 1.1 del software, esto resulta bastante lógico ya que ahí se encuentras los mayores tiempos de respuesta del sistema.

Versión	Nombre TRAZA	Tiempo Promedio	Tiempo Total	Cantidad de Request	Cantidad de Consultas a la BD
Versión 1.0	Escenario Actual Log 24 Meses	2299 ms	441580 ms	192	215
Versión 1.1	Optimización Datos	266 ms	50206 ms	192	215
Versión 1.2	Optimización Negocio	245 ms	47964 ms	195	109
Versión 1.3	Optimización Presentación	266 ms	47694 ms	179	109

Tabla 28 - Detalle resumen ejecuciones; v1.0, v1.1, v1.2, v1.3

Lo anterior quiere decir que la creación de índices que realizamos para los métodos que se identificaron utilizando el Profiler Multidimensional, fue suficiente para mejorar los tiempos de respuesta del aplicativo. Por otra parte la solución de implementar Caché, en la capa de negocio, para los datos que eran invariables y altamente utilizados, nos permitió disminuir los accesos a la base de datos, si bien los tiempos de respuesta no disminuyeron en la misma proporción que en la versión 1.1 del software, en la versión 1.2 del software la disminución de la cantidad de accesos a la base de datos nos permitió descongestionar dicho recurso aumentando así su disponibilidad. Y por último la mejora

en la interfaz de usuario, disminuyó a los request en el servidor web, lo anterior nuevamente no nos mejoró los tiempos de respuestas, tampoco los empeoró, pero si liberó recursos para poder atender a más usuarios de forma simultánea.

Con el fin de corroborar nuestra hipótesis que indica que con el paso del tiempo el sistema podría no adaptarse a las nuevas condiciones del entorno, decidimos comparar nuestro software, en nuestro escenario actual (24 meses de uso del sistema), que nos presenta graves problemas de performance, con un escenario anterior (5 meses de uso del sistema), el cual no tenía problemas de performance. En la tabla siguiente se puede ver como los tiempos de respuesta aumentaron en cerca de un 300%, lo cual nos indica que con la misma versión del software pero con menor carga el sistema, se comportaba de manera óptima, pero con el cambio de las condiciones, en este caso la carga constante de información, pone en evidencia los problema de diseño, sin embargo no nos permite identificar cual es el problema, aunque si nos permite tener una pista ya que el comportamiento de nuestro profiler multidimensional en ambos casos es prácticamente igual(visualmente), variando solo en los tiempos de respuesta, lo cual podría suponerse que si aplicamos las mismas reglas de optimización sobre nuestro software podríamos mejorar los tiempos, sin embargo con 5 meses de uso el sistema no presentaba ningún síntoma de problemas de performance.

Versión Software	Nombre TRAZA	Tiempo Promedio	Tiempo Total	Cantidad de Request	Cantidad de Consultas a la BD
Versión 1.0	Log 24 Meses	2299 ms	441580 ms	192	215
Versión 1.0	Log 5 Meses	738 ms	143991 ms	192	215

Tabla 29 - Detalle resumen ejecución v1.0 con 5 meses de carga versus 24 Meses de carga

Para poder corroborar nuestros resultados, generamos un nuevo experimento, en el cual decidimos potenciar el hardware que ocupa el sistema, con el fin de ver si realmente con un aumento de recursos (RAM y CPU) podríamos de manera rápida mejorar los tiempos de respuesta del sistema, y si bien los tiempos disminuyeron en cerca de un 10%, al lado de los resultados de las optimizaciones no podemos decir que la mejora fue significativa. Esto se debe a que los problemas de rendimiento se debían a un problema en la capa de datos del sistema, y no a problemas de infraestructura, probablemente si se hubiese mejorado el storage del sistema, los cambios habrían sido mejores, pero aun así difícilmente podríamos mejorar los tiempos del sistema optimizado.

Versión Software	Nombre TRAZA	Tiempo Promedio	Tiempo Total	Cantidad de Request	Cantidad de Consultas a la BD
Versión 1.0	Hardware Actual Log 24 Meses	2299 ms	441580 ms	192	215
Versión 1.0	Hardware Potenciado Log 24 Meses	2065 ms	396622 ms	192	215

Tabla 30 - Detalle resumen ejecución v1.0 con hardware actual ver Hardware Potenciado

Finalmente podríamos decir que para poder optimizar un sistema utilizando el Profiler Multidimensional descrito en este estudio, la creación del modelo es fundamental, ya que este nos debe permitir no solo identificar los cuellos de botella, si no que mediante

el uso de dimensiones nos permitan identificar problemas típicos, los cuales se resuelven con patrones de diseño u optimizaciones ampliamente conocidas. Por qué entonces no optimizar el sistema completo antes de sacar una versión 1.0 o productiva, simple, porque no sabemos a priori de que forma el sistema podría verse afectado por los cambios externos/internos que podrían hacer que nuestro diseño original no sea el óptimo para este nuevo escenario.

Es importante recalcar que para este estudio el modelo que se utilizó fue pensado para el software a analizar, si bien es cierto, puede extrapolarse a otros sistemas, es relevante que el análisis de dimensiones se haga pensando en el software que debemos optimizar, por ejemplo, para otro modelo arquitectónico podría no aplicar los mismos cuellos de botellas y mejoras propuestas, sin embargo, el Modelo y el Profiler Multidimensional se puede adaptar al software que se requiera optimizar. Por otra parte las dimensiones se pueden expresar visualmente de muchas formas, como vimos en el estudio, podríamos utilizar una circunferencia u otros colores en caso de cierta situación que nos de otra oportunidad de mejora.

BIBLIOGRAFÍA

1. *Andrew Black, Stéphane Ducasse, Oscar Nierstrasz, Damien Pollet, Damien Cassou, and Marcus Denker. Pharo by Example, Square Bracket Associates, 2009.* http://pharobyexample.org

2. *Tudor Girba, The Moose Book, Moose Version 4, http://www.themoosebook.org/book*

3. *Julio Ariel Hurtado Alegria, Alejandro Lagos, Alexandre Bergel and Maria Cecilia Bastarrica -- Software Process Model Blueprints. Proceedings of the International Conferences on Software Processes (ICSP'10)*

4. *Bendraou, R., Jezéquél, J.-M., Fleurey, F.: Combining Aspect and Model-Driven Engineering Approaches for Software Process Modeling and Execution. In: Wang, Q., Garousi, V., Madachy, R., Pfahl, D. (eds.) ICSP 2009. LNCS, vol. 5543, pp. 148–160. Springer, Heidelberg (2009).*

5. *Cook, J.E., Wolf, A.L.: Software process validation: quantitatively measuring the correspondence of a process to a model. ACM TOSEM 8(2), 147–176 (1999).*

6. *Ge, J., Hu, H., Gu, Q., Lu, J.: Modeling Multi-View Software Process with Object Petri Nets. In: ICSEA 2006, p. 41 (2006)*

7. Juan Pablo Sandoval Alcocer, Alexandre Bergel, Stephane Ducasse, Marcus Denker, *"Performance evolution blueprint: Understanding the impact of software evolution on performance", 2013/9/27.*

8. ***Sape*** research group at the Faculty of Informatics of the University of Lugano http://sape.inf.usi.ch/vertical-profiling