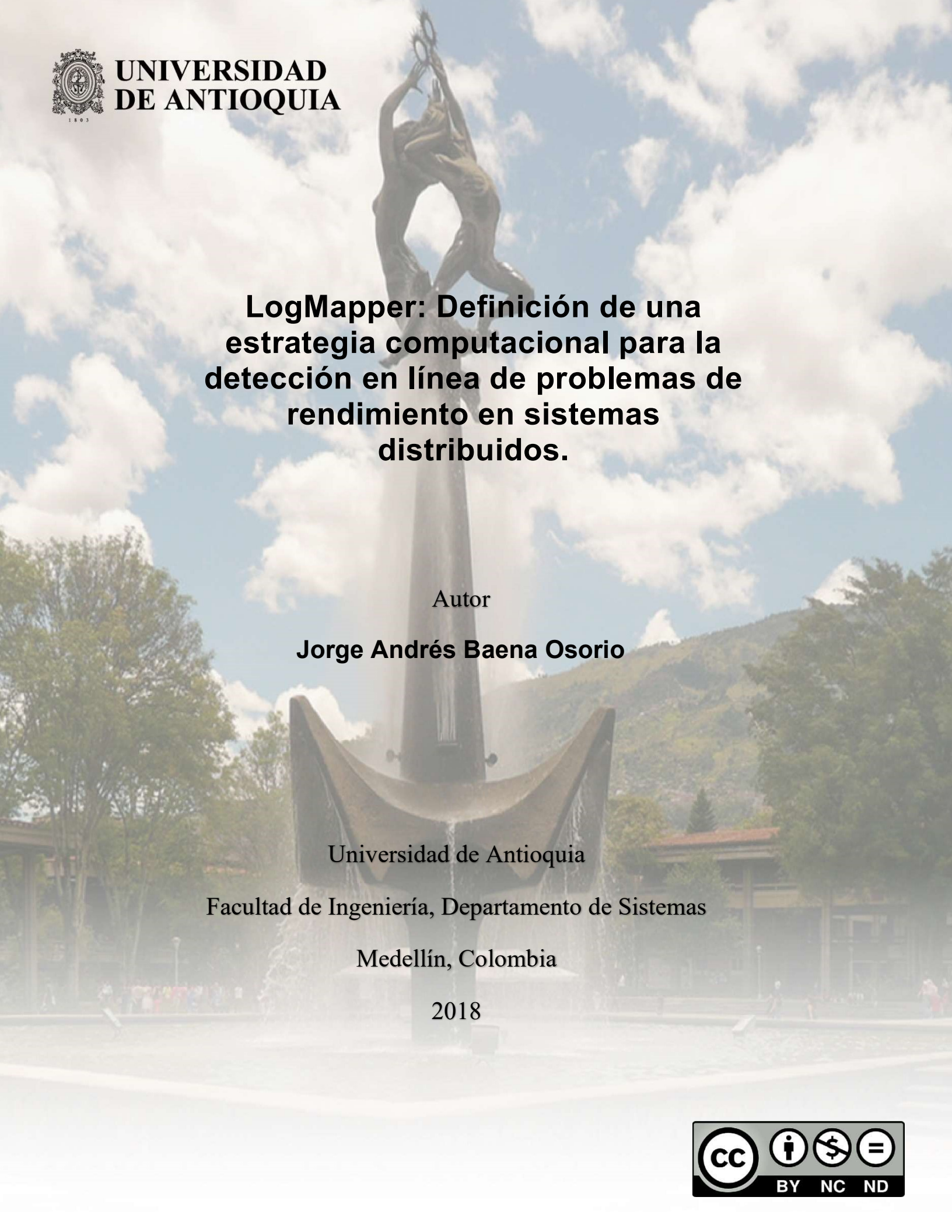


UNIVERSIDAD DE ANTIOQUIA

1803

LogMapper: Definición de una estrategia computacional para la detección en línea de problemas de rendimiento en sistemas distribuidos.

Autor

Jorge Andrés Baena Osorio

Universidad de Antioquia

Facultad de Ingeniería, Departamento de Sistemas

Medellín, Colombia

2018

LogMapper: Definición de una estrategia computacional para la detección en línea de problemas de rendimiento en sistemas distribuidos

Jorge Andrés Baena Osorio

Trabajo de investigación presentado
como requisito para optar al título de:
Maestría en Ingeniería.

Director:

PhD. John Freddy Duitama Muñoz

Universidad de Antioquia
Facultad de Ingeniería, Departamento de Sistemas
Medellín, Colombia
2018

A mi familia, por toda su paciencia y apoyo.

Agradecimientos

Agradezco especialmente a OSP International por su apoyo para realizar este Magister y por facilitar sus productos para verificar la herramienta propuesta. También a todos los compañeros que colaboraron con sus sugerencias y revisiones.

Resumen

El rendimiento es un atributo de los sistemas informáticos que hace referencia a la capacidad de entregar una respuesta en un tiempo determinado. Los efectos de un problema de rendimiento son indisponibilidad del servicio, rechazo de los usuarios, retraso en la operación y sobrecostos. Una necesidad de las organizaciones que utilizan o desarrollan sistemas distribuidos es poder detectar, en línea, la causa raíz de este tipo de problemas.

El presente trabajo expone una estrategia computacional usada para recopilar información desde todos los componentes involucrados en el procesamiento de un sistema distribuido, medir el rendimiento a partir de los archivos log y, utilizando técnicas de aprendizaje de máquinas, predecir si el comportamiento del rendimiento es el típico del sistema o si, por el contrario, se trata de una anomalía.

Se desarrolla una herramienta denominada *LogMapper* que, gracias a una arquitectura distribuida, puede capturar y procesar en línea la información mencionada, para poder ofrecer reportes gráficos en línea, que permiten detectar y diagnosticar, fácil y rápidamente, problemas de rendimiento.

Dicha estrategia ofrece una alternativa novedosa para diagnosticar el rendimiento con respecto a los trabajos previos, ya que involucra la duración de los flujos de operación, las métricas de los nodos y componentes y técnicas de aprendizaje de máquinas.

Palabras clave: rendimiento de software, aprendizaje de máquinas, sistemas distribuidos, archivos de log, microservicios, SOFTWA.

Abstract

Performance is an attribute of computer systems that refers to the ability to get a response between a certain amount of time. The effects of a performance problem are unavailability of service, user rejection, delay in operation and overrun cost. A need for organizations that use or develop distributed systems is to be able to detect, online, the root cause of a performance problem.

This project presents a computational strategy which information is collected from all the components involved in the processing of a distributed system, measures performance based on log files and, using machine learning techniques, predicts if it is a typical performance or if, in contrary, it is an anomaly.

A tool called LogMapper was developed, which is supported in a distributed architecture, can capture and process in line the mentioned information in a way that allows the generation of graphical in-line reports which can detect and diagnose, easily and quickly, performance issues.

That strategy offers a novel alternative to diagnose performance in relation to previous works, because it involves the duration of the operation flows, nodes and components' metrics and machine learning techniques.

Keywords: Performance, machine learning, distributed systems, log files, microservices, big data.

Contenido

Pág.

Lista de figuras

Pág.

Lista de tablas

Pág.

Lista de abreviaturas

Abreviaturas

Abreviatura	Término
CSV	*Comma-separated values*. Valores separados por coma
GCE	*Google Compute Engine*
GCP	*Google Cloud Platform*.
ML	*Machine Learning*. Aprendizaje de maquinas
OCSVM	*One Class Support Vector Machine*.
RMSE	*Root mean square error*. Error cuadrático medio.
SD	Sistema distribuido
SM	Siplex Management.
SVM	*Support vector machine*. Máquina de soporte vectorial
UI	*User interface*. Interfaz de usuario
URL	*Uniform Resource Locator*.
VM	*Virtual Machine*. Máquina virtual

1. Introducción

El rendimiento (*performance*) es un atributo o característica de calidad del software que hace referencia a la capacidad de un sistema de dar una respuesta determinada en un tiempo previamente definido. En ingeniería de software el rendimiento es un requerimiento no funcional que se refiere a velocidad, eficiencia, consumo de recursos, capacidad de procesamiento y tiempo de respuesta [1][2]. En la vida diaria ocurren problemas de rendimiento cada vez que se nota que algún equipo o software esta “lento”.

Los problemas de rendimiento son un asunto relevante en un número significativo de proyectos (al menos 20% según un estudio del año 2006 [3]), al punto, que de no ser manejados adecuadamente, pueden llevar al fracaso de todo el proyecto. Las consecuencias directas de un problema de rendimiento son: indisponibilidad del servicio, rechazo de los usuarios, retraso en la operación y sobrecostos[4].

Los sistemas distribuidos son aquellos que están conformados por múltiples componentes, cada uno de los cuales cumple una función determinada. Dichos componentes están normalmente en computadores distribuidos a través de una red y coordinan sus acciones a través del intercambio de mensajes. Aunque no los vemos directamente, los sistemas distribuidos hacen parte de nuestra vida cotidiana por medio de los servicios en la nube tales como Google, Netflix, Amazon, Dropbox, etc. En un sistema distribuido la ejecución de una operación implica un flujo de operaciones y mensajes entre múltiples nodos. El tiempo de respuesta total de dicha operación está directamente relacionado con los tiempos de respuesta de cada componente implicado en la transacción. Por lo tanto, el deterioro del rendimiento de un nodo afecta otros nodos y afecta el tiempo de respuesta global del sistema.

Cuando en un sistema distribuido se presenta una falla de rendimiento, se requiere que los administradores del sistema identifiquen rápidamente las causas del problema para que puedan implementar una solución en corto tiempo. La causa raíz incluye identificar los nodos cuyos tiempos de respuesta son altos, o están fuera de lo habitual, y posteriormente identificar dentro del componente la clase y método que están presentando la falla. En ocasiones la causa raíz no se debe a fallos intrínsecos del software si no a factores externos, tales como el estado del hardware o la velocidad de la red.

Los archivos de log generados automáticamente por el sistema son la fuente primaria para diagnosticar la causa raíz del error, buscando identificar registros de excepciones, u operaciones que presenten algún tipo de comportamiento o estado anormal. El proceso de búsqueda y análisis requiere que la persona asignada a dicha tarea tenga un conocimiento profundo del sistema completo, que conozca la arquitectura general, lo diferentes componentes del sistema, los mensajes entre componentes y sus correspondientes secuencias. Solo así podrá identificar de una manera eficiente y eficaz cuándo un comportamiento debe ser tomado como sospechoso o cuando se trata de una operación normal.

El proceso de diagnóstico anterior conlleva dos problemas que aumentan en complejidad a medida que el sistema distribuido crece, primero, la cantidad de información a analizar es muy alta, y es complejo seguir la secuencia de mensajes y definir la relevancia de cada mensaje. El segundo problema, es que muchas veces no se cuenta con la persona con la suficiente experticia y conocimiento del sistema distribuido para realizar el diagnóstico, ya que inclusive en muchas ocasiones los grupos de desarrollo trabajan de manera independiente [5].

Dado todo lo anterior se podría sintetizar la definición del problema de la siguiente manera: Dado el tamaño y complejidad de los sistemas distribuidos se requiere recurrir a estrategias computacionales que apoyen de manera eficiente y eficaz la detección en línea de problemas de rendimiento en dichos sistemas. Dichas estrategias deben permitir “aprender” de manera automática el comportamiento normal del sistema, para que al procesar en línea la información de los archivos log tenga la capacidad de detectar problemas de rendimiento, identificando los nodos afectados y ofreciendo información que permita identificar la causa raíz del problema.

El presente trabajo expone una estrategia computacional usada para recopilar información desde todos los componentes involucrados en el procesamiento de un sistema distribuido y, utilizando técnicas de aprendizaje de máquinas, diagnosticar fallas de rendimiento. Dicha estrategia mide un indicador del rendimiento utilizando los archivos log de la aplicación. Para lograrlo infiere los flujos de operación del sistema y su duración típica, la cual es utilizada como valor de referencia para medir el rendimiento. Adicionalmente, se captura información relacionada a los componentes y los nodos del sistema distribuido, compuesta por métricas de carga y contadores de eventos tales como errores o advertencias. Toda la información recopilada es procesada para aprender el comportamiento típico del sistema, utilizando la técnica OCSVM (*One Class Support Vector Machine*) para detección de anomalías y regresión lineal multivariada para calcular un valor estimado del indicador del rendimiento.

Se desarrolla una herramienta denominada *LogMapper* que, gracias a una arquitectura distribuida, puede capturar y procesar en línea la información mencionada, para poder ofrecer reportes gráficos en línea, que permitan detectar, fácil y rápidamente, problemas de rendimiento identificando su causa raíz.

Los beneficios del uso de *LogMapper* para las organizaciones que utilizan sistemas distribuidos son:

- Disminución del tiempo de afectación del servicio en sistemas productivos debido a fallas de rendimiento.
- Mejoras en el índice de disponibilidad del servicio.
- Mejora en la percepción de los usuarios.
- Ahorros económicos por la mayor disponibilidad de servicio.

Los beneficios para las organizaciones que desarrollan software distribuido son:

- Disminución del tiempo de desarrollo de aplicaciones de software.
- Incremento de la calidad en el desarrollo de software.
- Disminución de tiempo de depuración debido a fallas de rendimiento.

1.1 Objetivo general

Desarrollar una herramienta de software que apoyada en técnicas de inteligencia computacional permita diagnosticar en línea problemas de rendimiento en sistemas distribuidos a partir de los archivos de log.

Dichas técnicas serán usadas para “aprender” de manera automática el comportamiento típico del sistema, y para que al procesar la información de los archivos log se tenga la capacidad de detectar problemas de rendimiento, identificando los nodos en falla y ofreciendo información que permita identificar la causa raíz del problema.

1.2 Objetivos específicos

1. Diseñar un modelo para la detección de problemas de rendimiento que permita caracterizar nodos, secuencias y duración y sea fácilmente adaptable a diferentes tipos de sistemas computacionales.
2. Diseñar y construir la herramienta de software que implemente el modelo de detección definido.
    a. Implementar módulo de captura de archivos que permita centralizar de manera eficiente la información relevante en un repositorio centralizado.
    b. Implementar módulo de preprocesamiento de archivos que permita procesar los archivos de log originales y convertirlos en una nueva estructura más fácil de analizar.
    c. Implementar módulo de creación de características que permite crear el modelo con los flujos y las características de duración.
    d. Implementar módulo de análisis de patrones que utiliza técnicas de aprendizaje de máquina para identificar si el comportamiento del sistema corresponde al comportamiento típico o se trata de un problema de rendimiento.
    e. Implementar módulo de generación de reportes que permita visualizar de una manera intuitiva y directa el comportamiento del sistema a nivel de rendimiento y en caso de falla visualizar los segmentos de log que presentan la anomalía.
3. Realizar la validación del modelo:
    a. Entrenar el modelo para que “aprenda” el comportamiento normal del sistema.

b. Ejecutar la herramienta y validar que se procesa en línea los archivos log del sistema entregando reportes del resultado del procesamiento de datos.
c. Provocar fallas en el sistema distribuido y validar que se detectan correctamente y en línea.

1.3 Estructura del documento

Este documento está estructurado de la siguiente forma: El capítulo 2 muestra los resultados de la revisión bibliográfica del estado del arte; en el capítulo 3 se hace una exposición del marco teórico que será utilizado en el desarrollo de la herramienta. En el capítulo 4 se hace la definición del modelo propuesto para dar solución al problema planteado, la cual está compuesta por tres partes principales:

1. Medición del rendimiento: se explica la metodología para medir el rendimiento a partir del procesamiento de los archivos log.
2. Aprendizaje de máquinas: se describen las técnicas utilizadas para realizar el aprendizaje para detectar anomalías y predecir el rendimiento, y se exponen los resultados de la validación del modelo entrenado.
3. Procesamiento en línea: se definen las estrategias utilizadas para poder hacer un procesamiento de datos que entregue resultados en línea.

El capítulo 5 presenta la información técnica relacionada con el diseño e implementación de la herramienta, el capítulo 6 muestra los resultados obtenidos y finalmente en el capítulo 7 se exponen las conclusiones y recomendaciones.

En los anexos se puede consultar información adicional de la implementación de la herramienta.

2. Estado del arte

Para determinar el estado del arte se realizó la revisión bibliográfica en cuanto a metodologías para evaluar el rendimiento de sistemas distribuidos utilizando archivos de log. Se hizo una revisión sistemática de artículos científicos en las bases de datos bibliográficas Scopus, IEEE Explore, ACM Digital Library, Science Direct y Web of Science.

La búsqueda bibliográfica se estructura con las siguientes preguntas de investigación:

- ¿Qué metodologías se han utilizado para detectar problemas de rendimiento utilizando archivos log?
- ¿Las metodologías encontradas se pueden implementar en sistemas distribuidos?
- ¿Las metodologías encontradas entregan resultados en línea?

El número de artículos encontrado en una búsqueda preliminar era muy alto, por lo que se ajustó el criterio de búsqueda como se muestra en la **Figura 2-1**. De esta forma los resultados se limitaban a menos de 60 artículos, de los cuales se seleccionaron los que correspondían a herramientas o metodologías que buscaban detectar problemas de rendimiento en sistemas distribuidos.

Título, resumen o palabras clave contiene ("performance")
Y Título, resumen o palabras clave contiene ("logs")
Y Título, resumen o palabras clave contiene ("distributed systems ")
Y Título, resumen o palabras clave contiene ("analysis" O "detect" O "measure")
Y año publicación > 1999.

Figura 2-1: Criterios de búsqueda para bases de datos bibliográficas

A continuación, se describen los trabajos previos encontrados en orden cronológico. Para cada trabajo, se pone especial atención en los siguientes criterios de análisis:

- Medición del indicador del rendimiento.
- Recolección y preprocesamiento de datos.

- Diagnóstico del rendimiento.
- Visualización.
- Procesamiento en línea.
- Requiere intervenir o conocer la aplicación
- Validación.

2.1 *Netlogger* (2000)

Netlogger [6][7] propone una metodología que consiste en un kit de herramientas para el análisis de rendimiento en sistemas distribuidos. Con esta metodología pretende monitorear, bajo condiciones reales de operación, el comportamiento de todos los elementos de comunicación dentro de la aplicación para determinar exactamente qué está pasando dentro de un sistema complejo. En esta solución se requiere generar un log con un formato determinado, por lo que ofrecen una librería para generar su formato propio de log. El formato utilizado por *Netlogger* usa el estándar ULF [8] (*Universal Logger Message format*) de la IETF. Este formato consiste en una lista de pares "campo" = "valor". El estándar define como campos obligatorios DATE, HOST, PROG y LVL. Netlogger adiciona el campo NL.EVNT que es un identificador único para el evento que se está monitoreando (Ver **Figura 2-2**).

```
DATE=19980430133038
HOST=dpss1.lbl.gov
PROG=testprog
LVL=Usage
NL.EVNT=SEND_DATA
NL.SEC=893968238
NL.USEC=55784
SEND.SZ=49332
```

Figura 2-2: Ejemplo registro de log ULF utilizado por Netlogger

Netlogger está compuesto por varios elementos: una librería llamada *Netlogger API* que debe ser incluida en el sistema que se va a monitorear. Esta librería permite generar los registros de log en el formato especificado. Un servidor denominado *netlogd* que recibe en un determinado puerto TCP registros de log de cualquier programa en la red y lo guarda en un archivo con el formato ULF. Otro componente es el recolector en tiempo real, el cual realiza tres funciones, recibe datos de la aplicación, procesa las peticiones remotas y transforma los formatos de log. Por último, está la herramienta de visualización llamada NLV (*Netlogger Visualization Tool*). Esta aplicación permite visualizar el comportamiento

del rendimiento del sistema con tres gráficas básicas: la línea de vida (*lifeline*), línea de carga (*loadline*) y puntos (*points*). En la **Figura 2-3** se puede observar un ejemplo de dichas gráficas.

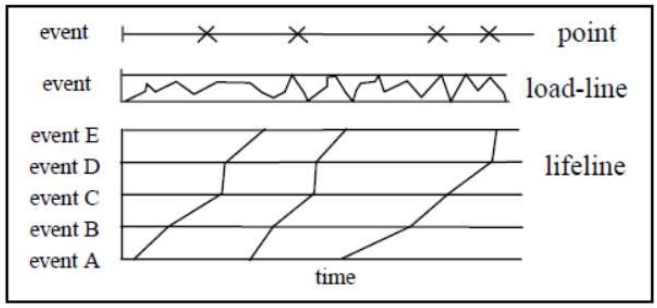


Figura 2-3: Visualización de resultados con *Netlogger*

La línea de vida representa la vida de un objeto a través de un sistema distribuido. La pendiente de las líneas de esta gráfica indica claramente la latencia entre las acciones de los sistemas. La línea de carga permite identificar los cambios en variables como CPU y memoria. La línea de puntos marca eventos en puntos específicos del tiempo, por ejemplo, errores del sistema. La metodología no entrega un valor de rendimiento, ni tampoco un diagnóstico especifico, los reportes entregados permiten a un usuario experto realizar un análisis del rendimiento del sistema distribuido.

La herramienta *Netlogger* permite hacer análisis de rendimiento en tiempo real, permitiendo procesar hasta 24000 mensajes de log por segundo. Para verificar la efectivad detectando problemas de rendimiento se aplicó la herramienta en una aplicación llamada *Radiance,* donde efectivamente se encontró el problema de rendimiento.

2.2 Depuración de rendimiento de cajas negras (2003)

En [9] los autores proponen una metodología para hacer depuración del rendimiento en sistemas distribuidos asumiendo los componentes como cajas negras. Este último supuesto les permite hacer el análisis utilizando los archivos log generados por cualquier sistema, sin necesidad de conocer su funcionamiento especifico, tampoco requiere que el log tenga un formato determinado. Su metodología busca determinar los patrones de los

flujos de ejecución, para identificar los que tengan un alto impacto en el rendimiento del sistema y dentro de estos ubicar los nodos que presentan una latencia mayor. Esta propuesta tiene tres características principales:

- Requerir mínimo conocimiento de la semántica de la aplicación.
- No requiere modificar la aplicación
- No debe perturbar de manera significativa la operación del sistema.

En esta aproximación los autores utilizan tres fases:

- Trazado de la comunicación: se identifican los mensajes entre nodos y se obtiene la traza de toda esta información. Este procedimiento se debe hacer en vivo, utilizando los archivos log de la aplicación.
- Inferir flujo de operación y sus patrones (Ver **Figura 2-4**): se utilizan algoritmos para determinar las secuencias de una operación. Esta fase es fuera de línea.
- Visualización: en esta fase se presentan los resultados de manera que sean fácil de entender para una persona.

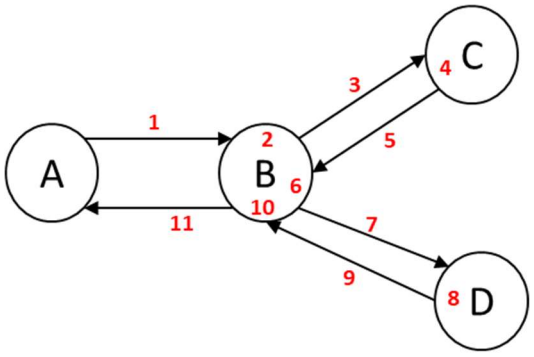


Figura 2-4: Ejemplo flujo de operaciones en sistema distribuido.

Para detectar los patrones de las secuencias de operación se utilizan dos algoritmos que solo requieren las estampas de tiempo y criterios estadísticos simples (frecuencia) para inferir causalidad. Un algoritmo que llaman de anidación y otro llamado de convolución.

El algoritmo de anidación ("*nesting algorithm*") combina todas las trazas por nodo y verifica cada traza individual para identificar qué llamados están anidados dentro de otro. En el ejemplo de la **Figura 2-5** el llamado de B a C y el llamado de B a D están anidados dentro

del llamado de A a B. Este tipo de análisis requiere que siempre exista una respuesta a un llamado. El algoritmo calcula posibles anidamientos utilizando las estampas de tiempo de los mensajes. Haciendo análisis estadísticos simples se infieren los anidamientos más probables para crear las relaciones de causalidad que se quieren detectar.

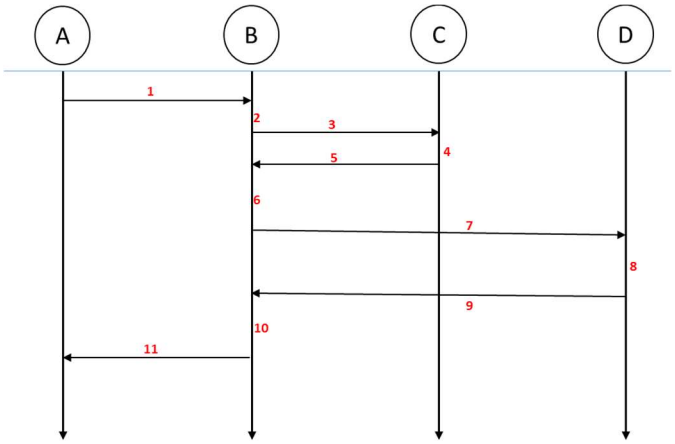


Figura 2-5: Algoritmo de anidación.

El algoritmo de convolución busca las relaciones de causalidad utilizando la agregación de múltiples mensajes. La idea es que el algoritmo toma un conjunto de trazas de log, y toma cada traza como una señal de tiempo, a las que se le aplica técnicas de procesamiento de señales para detectar las correlaciones entre estas. Este algoritmo no requiere que todos los mensajes tengan respuesta, ya que toma todas las señales de manera separada. En la **Figura 2-7** se muestra la representación gráfica del algoritmo de convolución. El eje X representa la ventana de tiempo y el eje Y el número de mensajes.

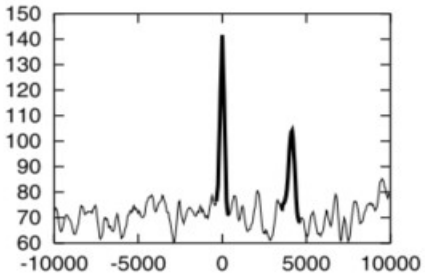


Figura 2-6: Algoritmo de convolución.

La visualización de la herramienta utiliza un programa externo llamado dot [10] que permite crear gráficas a partir de un lenguaje de texto. Las gráficas resultantes de esta solución son grafos como los mostrados en la **Figura 2-7**. La metodología se verificó con varias aplicaciones en experimentos controlados, obteniendo resultados útiles para encontrar las relaciones entre los llamados de diferentes componentes y medir su latencia. Un usuario experto debe determinar si los tiempos de latencia detectados son normales o si son síntoma de un problema de rendimiento.

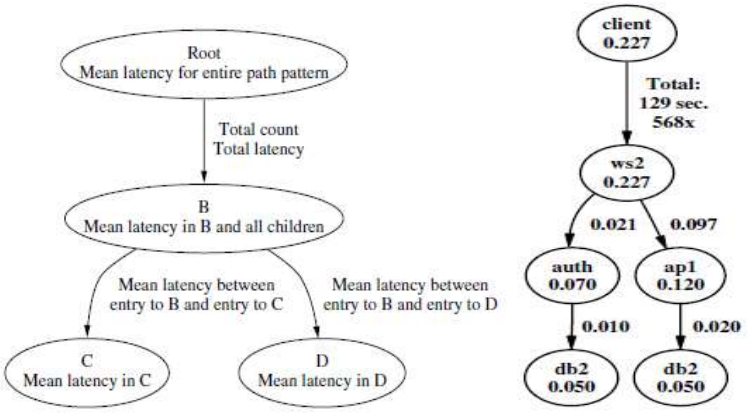


Figura 2-7: Formato y ejemplo de salida del análisis de rendimiento.

2.3 WAP5 (2006)

WAP5 (*Wide-Area Project 5*) [11] es un proyecto que busca hacer más fácil las tareas de depuración, optimización y mantenimiento de los sistemas distribuidos de área amplia por medio de inferencia de las relaciones de causalidad y de los tiempos de latencia de las comunicaciones del sistema. Los retardos encontrados permiten evidenciar los cuellos de botella que afectan el rendimiento del sistema distribuido.

Este proyecto está relacionado con el trabajo de depuración de rendimiento de cajas negras (Capítulo 2.2) pero se extiende a sistemas distribuidos de área amplia donde entran nuevas fuentes de retardos y fallas tales como latencia de red, ancho de banda limitado, fallas de nodos y programación paralela. La herramienta está compuesta por cuatro módulos tal como se muestra en la **Figura 2-8**. Para capturar la traza el sistema genera logs a través de un componente que captura la información de las peticiones de red. Este

componente fue desarrollado específicamente para esta herramienta para capturar el host, el proceso, el destino, el tamaño del paquete y la estampa de tiempo.

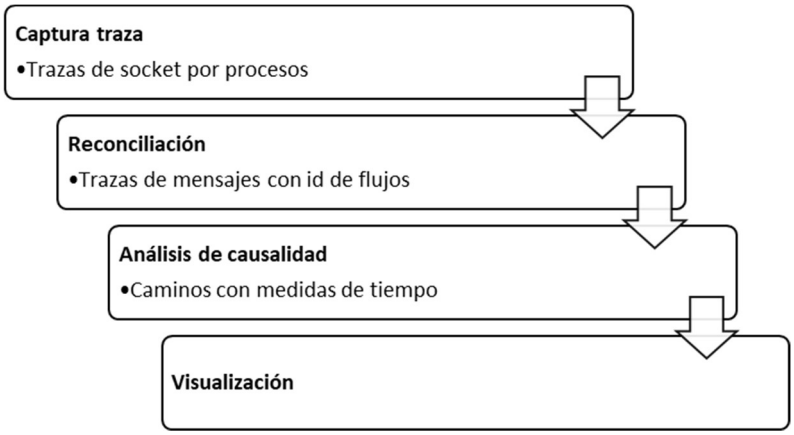


Figura 2-8: Módulos y proceso general de WAP5.

La salida de esta herramienta es una representación gráfica que permite observar los flujos de peticiones y los tiempos de latencia entre los diferentes llamados (Ver **Figura 2-9**). Esta información puede ser analizada para determinar los problemas de rendimiento. La herramienta fue verificada en dos sistemas distribuidos (*CoDeeN y Coral PlanetLab*) obteniendo resultados positivos. Un usuario experto debe determinar si los tiempos de latencia detectados son normales o si son síntoma de un problema de rendimiento.

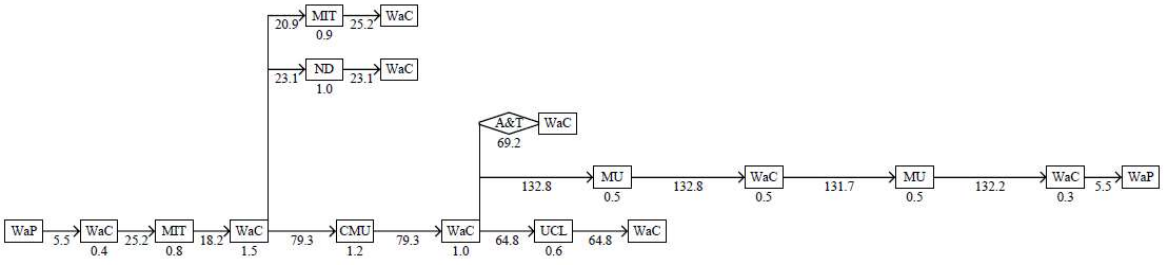
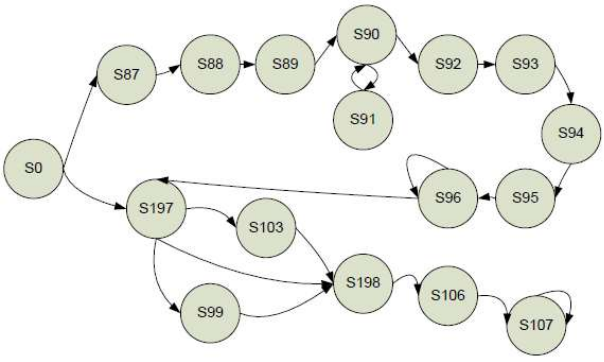


Figura 2-9: Ejemplo resultados WAP5.

2.4 Detección de anomalías de ejecución a través de análisis de log (2009)

Esta aproximación [12] propone un algoritmo que convierte mensajes log no estructurados en objetos denominados *log keys*. Posteriormente se utiliza un autómata de estado finito (FSA *Finite State Automaton*) para aprender el flujo normal por cada componente del sistema. Ya obtenidos los modelos de los componentes del sistema se pueden utilizar para detectar anomalías en nuevos archivos de log.

Los *log-key* corresponden a las líneas de log obtenidas en los archivos log. Dado que la mayoría de las veces los archivos log no tienen estructura y pueden tener datos que cambian constantemente, el número de mensajes diferentes en los archivos log puede ser muy alto. Para minimizar este problema se hace un procesamiento sobre las líneas de log, donde se eliminan parámetros por reglas empíricas y se hace un proceso de agrupamiento para obtener un número de *log-keys* mucho menor. Para modelar el flujo de operación del sistema se utiliza un autómata de estado finito, el cual consiste en un número finito de estados y transiciones entre estados. Cada transición corresponde a un *log-key*, y se interpreta como un cambio en el estado operacional del sistema. El algoritmo utilizado para crear la máquina de estado finito está definido en [13]. En la **Figura 2-10** se muestra un ejemplo de un autómata de estado finito del módulo *JobTracker* de *Hadoop* inferido automáticamente, sin embargo, se debe aclarar que la interpretación de los estados se debe hacer manualmente.



State	Interpretation
S87~ S96	Initialization when a new job submitted
S197	Add a new map/reduce task
S103	Select remote data source
S99	Select local data source
S198	Task complete
S106	Job complete
S107	Clear task resource

Figura 2-10: Ejemplo autómata de estado finito para *Hadoop*

Para medir el rendimiento se definen dos tipos de problemas de rendimiento:

- El intervalo de tiempo para pasar de un estado al siguiente es mucho más largo de lo normal.
- Número de ciclos en un bucle es mucho mayor al caso normal.

Para representar la distribución del intervalo de tiempo para pasar de un estado a otro se utiliza un modelo Gaussiano, donde se tiene media y varianza. Para detectar las anomalías de un nuevo valor se calcula un umbral, el cual es comparado con el nuevo dato. Si supera el umbral hay una anomalía.

La técnica propuesta fue evaluada con Hadoop y un sistema distribuido privado llamado SILK, para los cuales se insertaron problemas de rendimiento de manera controlada y se verificó que se identificaran los estados que presentaban bajo rendimiento.

2.5 ASDF (2010)

ASDF (*Automated, Online Framework for Diagnosing Performance Problems*) [14] define una arquitectura flexible que permite a los administradores del sistema personalizar diferentes fuentes de datos y módulos de análisis a su ambiente particular para monitorear y analizar problemas de rendimiento especificando el nodo o el conjunto de nodos que presentan falla de rendimiento. Este *framework* define los siguientes componentes (Ver **Figura 2-11**):

- Recolector de datos: son demonios o servicios que corren en cada nodo y recolectan los datos de diferentes fuentes tales como logs de aplicación y variables de rendimiento del sistema operativo.
- Agregador de datos: periódicamente guardan los datos en un archivo persistente.
- Análisis de datos: periódicamente ejecuta un análisis de los datos para detectar problemas de rendimiento. El análisis puede correr localmente en cada nodo o globalmente en el nodo central. Se manejan umbrales para minimizar los falsos positivos.
- Almacenamiento persistente: lugar para almacenar datos históricos
- Alarmas y visualización: genera en línea alertas que indiquen los nodos en falla

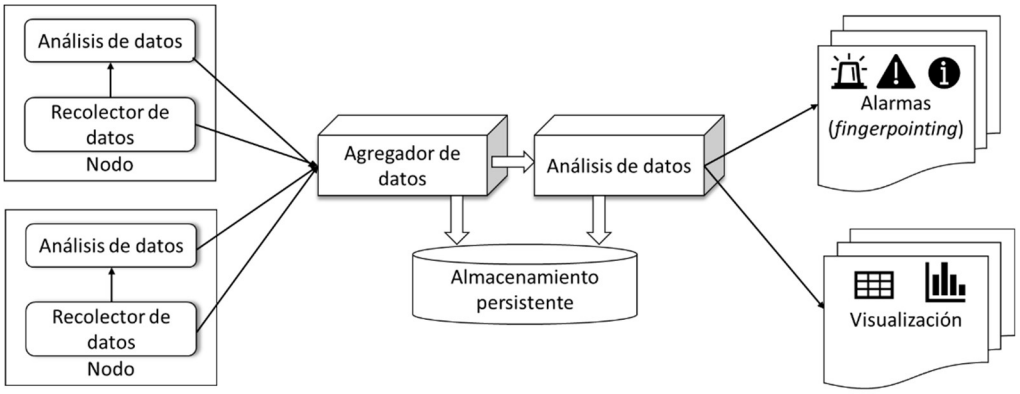


Figura 2-11: Componentes principales de ASDF.

La característica central del *framework* ASDF es la capacidad de incorporar cualquier número de fuentes de datos en un sistema distribuido y de agregar múltiples módulos de análisis de datos. Para lograr lo anterior el sistema encapsula las fuentes de datos y las técnicas de análisis en módulos, los cuales tienen definidos entradas y salidas y deben implementar un API que permite integrarse al módulo principal denominado *fpt-core* (*fingerpoint-core*).

Los módulos de fuentes de datos puedes ser tipo caja negra (*black box*) o caja blanca (*white box*). Caja negra se refiere a que el módulo no conoce el comportamiento de la aplicación, ni su estructura, y su objetivo es inferir su comportamiento extrayendo datos de forma transparente para ésta, sin utilización de instrumentos especiales. En los casos tipo caja blanca se extrae información de la capa de aplicación, requiere instrumentos específicos y normalmente se requiere conocimiento de la funcionalidad y de la estructura de la aplicación. La herramienta dispone de los siguientes módulos funcionales:

- sadc: capturas métricas del nodo: CPU, Memoria, I/O, tráfico de red, etc.
- hadoop_log: captura información propia de implementaciones Hadoop.
- mavgvec: módulo de análisis que calcula media y varianza de un vector correspondiente a una ventana de tiempo.
- knn: modulo de análisis para agrupar datos por medio del algoritmo k-vecinos (*k-nearest neighbors*).

Para demostrar la validez de la propuesta los autores utilizaron ASDF para localizar problemas de rendimiento en Hadoop, para lo cual utilizaron análisis de caja negra y caja

blanca y un *benchmark* especializado para *Hadoop* llamado *GridMix*. Los resultados de los experimentos realizados fueron de un valor de precisión balanceada entre 68% y 84%. Un valor alto de precisión balanceada indica una rata alta de verdaderos positivos y una tasa baja de falsos positivos.

2.6 Magnifier (2011)

Magnifier [15] es una herramienta para hacer la detección en línea de problemas de rendimiento en sistemas de computación en la nube. El flujo de ejecución es modelado por una estructura jerárquica por capas: componentes, módulos y funciones (Ver **Figura 2-12**). Un componente sería cada proceso ejecutable del sistema distribuido, cada componente está compuesto por uno o más módulos, por ejemplo, los paquetes de JAVA, y finalmente dentro de estos módulos están las clases con sus métodos o funciones.

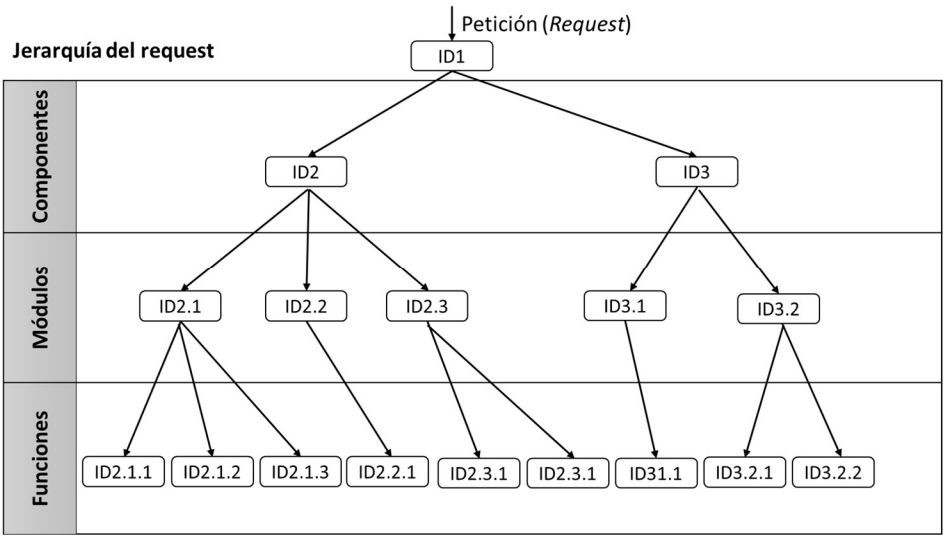


Figura 2-12: Estructura jerárquica de las peticiones en Magnifier.

Las anomalías son detectadas de la capa superior a la inferior independientemente. *Magnifier* primero compara el comportamiento de los componentes en el periodo actual con un periodo considerado normal. Si alguno presenta una degradación es detectado. Posteriormente verifica la capa de módulos, e identifica el *top* K de los módulos que más aportan a la degradación del rendimiento. Finalmente, después de aplicar un análisis de

componente principales, puede filtrar los métodos con menos influencia en el rendimiento y localizar los que están causando de raíz la falla de rendimiento.

Magnifier requiere la creación de trazas de log dentro de la aplicación, las cuales son enviadas a un proceso que se ejecuta en cada nodo. Para localizar los elementos con comportamiento anormal se comparan el promedio del tiempo de latencia y la fluctuación del periodo actual con un periodo conocido como normal.

La herramienta fue verificada realizando varios experimentos en la plataforma distribuida Alibaba, comprobando que *Magnifier* ofrece información que permite identificar la causa raíz de un problema de rendimiento.

2.7 Distalyzer (2012)

Distalyzer [5] es una herramienta que busca que los desarrolladores puedan analizar fácilmente temas relativos al rendimiento de sistemas distribuidos, sin tener que conocer el funcionamiento detallado de todos los componentes. Para lograrlo la herramienta utiliza técnicas de modelado descriptivo y predictivo para analizar archivos de log y extraer medidas de rendimiento. La solución construida compara dos conjuntos de archivos log, uno que corresponda a un buen rendimiento del sistema y otro con bajo rendimiento, y automáticamente identifica los eventos que afectan el comportamiento global del sistema en los logs de bajo rendimiento.

Distalyzer analiza los archivos log de una manera agnóstica, es decir, no requiere un formato específico, sin embargo, para que sea efectivo el formato del log debe permitir identificar dos categorías de mensajes:

- Mensajes de Eventos: Indica la ocurrencia de un evento en el momento que el log es generado. Estos mensajes son útiles para hacer trazado del control de flujo de ejecución entre diferentes componentes del sistema.
- Mensajes de estado: Indican el valor actual de alguna variable del sistema en el momento que se genera el log.

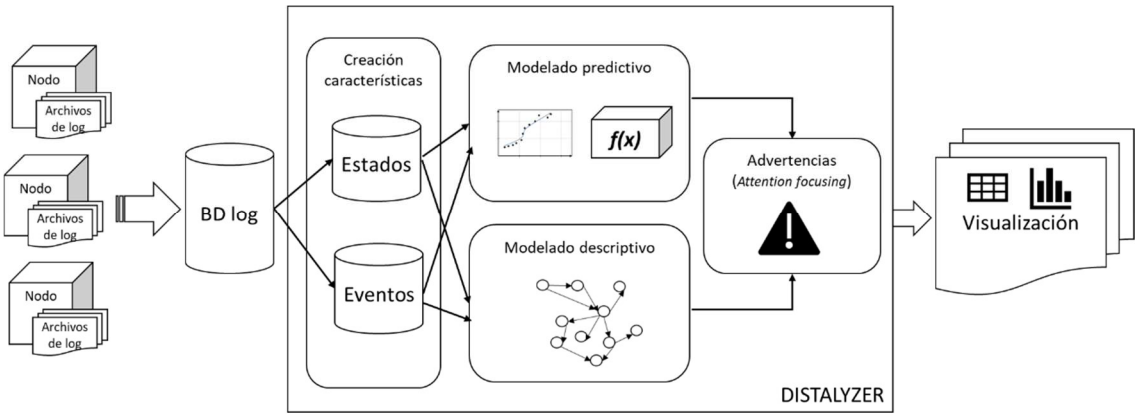


Figura 2-13: Etapas del análisis con *Distalyzer.*

En caso de que los mensajes de log no se ajusten claramente en estas categorías un primer componente de Distalyzer convierte todos los mensajes de log originales a un formato propio de mensajes de eventos y mensajes de estado. El flujo de procesamiento de Distalyzer se puede resumir en cuatro etapas que se resumen en la **Figura 2-13**.

- Creación de características: se extrae de los archivos log un pequeño conjunto de características tipo evento/estado para hacer más eficiente el análisis automático.
- Modelado Predictivo: las variables tipo evento/estado son analizadas con métodos estadísticos (Pruebas t-test de Welch, Kolmogorov-Smirnov) para identificar las diferencias entre los dos conjuntos de logs.
- Modelado Descriptivo: dentro de cada log se analizan las relaciones entre las variables evento/estado y se crea una red de dependencias.
- Puntos de Atención: los resultados de los pasos anteriores son combinados para generar una representación gráfica que indica donde debe enfocarse para analizar el problema de rendimiento. (Ver **Figura 2-14**)

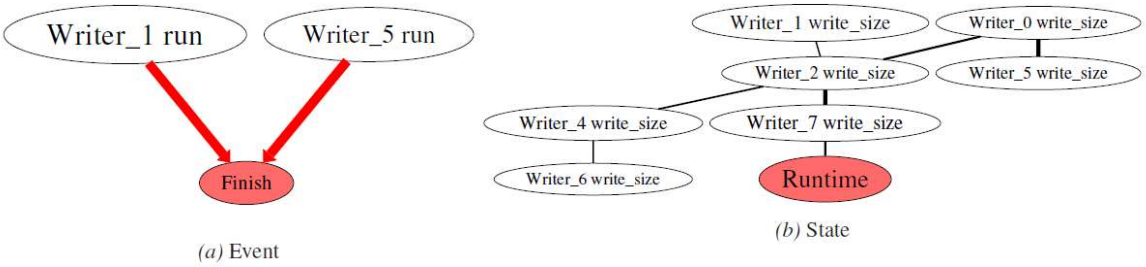


Figura 2-14: Grafo de dependencias obtenido con *Distalyzer*

La herramienta fue validada con tres casos de estudio: *TritonSort*, *Hbase* y *Transmission*, para los cuales se crearon log en condiciones normales y en fallas generadas.

2.8 Splunk (2014)

Splunk [16][17] es una aplicación comercial que recolecta e indexa datos en tiempo real de aplicaciones, servidores web, servidores de aplicaciones, bases de datos, redes cableadas, máquinas virtuales, dispositivos móviles, equipos de telecomunicaciones, sistemas operativos, sensores, y todo tipo de datos generados por máquinas para ofrecer diferentes funcionalidades tales como análisis de seguridad, monitoreo de la operación, detección de fallas y de patrones (Ver **Figura 2-15**).

Splunk tiene una interfaz de usuario sencilla pero muy funcional (Ver **Figura 2-16**), que permite agregar y configurar fácilmente diferentes tipos de fuentes de datos. La herramienta de consultas permite analizar los datos capturados, con un buscador que acepta expresiones complejas y operadores estadísticos. Los reportes se pueden configurar con diferentes tipos de visualizaciones, incluyendo tablas, gráficas y líneas de tiempo. También tiene un módulo de monitoreo y alertas, que analiza los datos en tiempo real y genera alertas al usuario cuando se detectan eventos que sobrepasan cierto umbral configurado.

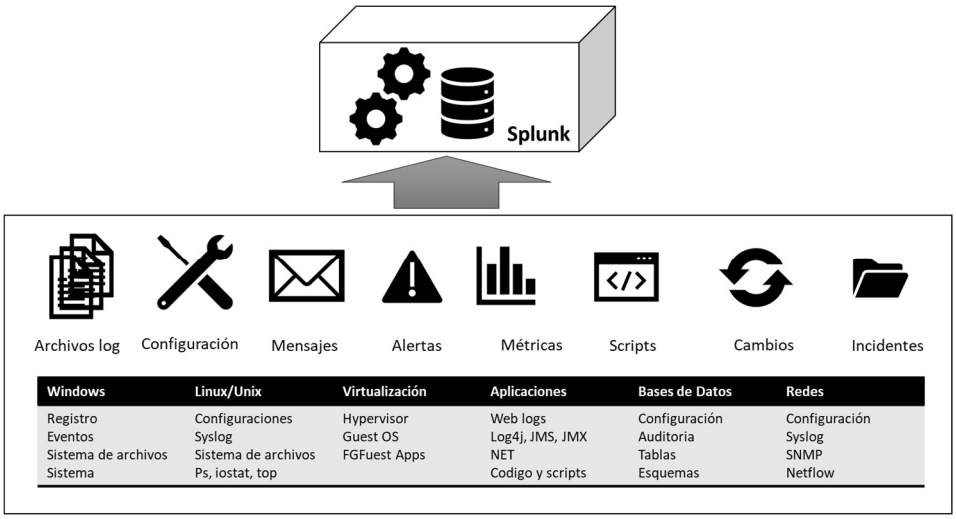


Figura 2-15: Fuentes de datos soportadas por *Splunk*[18]

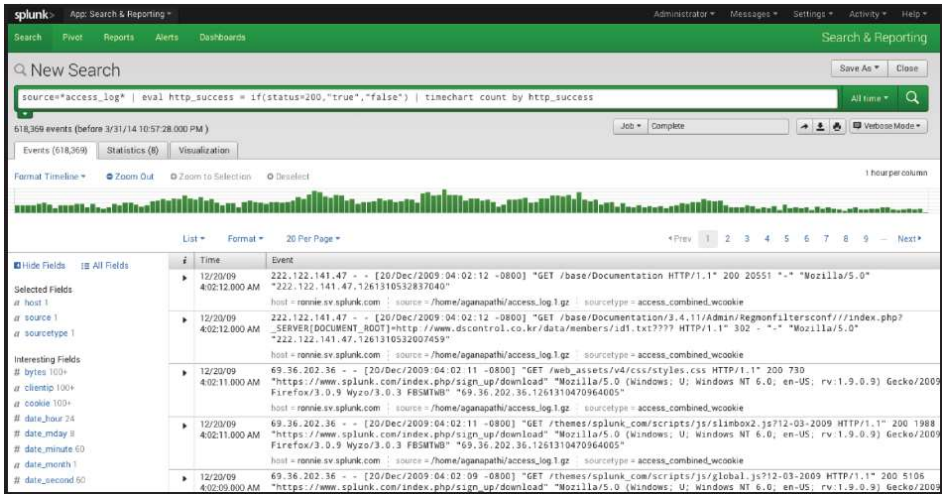


Figura 2-16: Interfaz gráfica de *Splunk*

Una de las características principales de *Splunk*, es la capacidad de procesar datos en tiempo real, para lograrlo soporta configuraciones escalables, lo que le permite implementar mecanismos tipo MapReduce en sistemas distribuidos. Los datos indexados recolectados pueden ser consultados a través de la interfaz de usuario usando una utilidad llamada *Splunk Query Language*, la cual soporta diferentes funciones típicas tales como agregación, creación de nuevas columnas, cache, agrupamiento, unión, filtrado entre otros. Esta última característica es la que permite configurar la herramienta para visualizar eventos específicos que permitan analizar el rendimiento. Sin embargo, crear los *queries*

adecuados requiere un conocimiento técnico particular del sistema distribuido, y la interpretación de la información obtenida depende del conocimiento de dicha persona.

2.9 MilliScope (2017)

MilliScope [19] es una herramienta que permite monitorear eventos y recursos para sistemas distribuidos para analizar problemas de rendimiento eventuales y de corta duración (del orden de milisegundos). El análisis del orden de milisegundos tiene relevancia ya que, según un reporte de Amazon, un incremento de la latencia de 100ms puede generar pérdidas hasta de un 1% en ventas. La herramienta no realiza un análisis de rendimiento especifico, si no que se especializa en capturar información con un nivel de granularidad suficiente para registrar eventos del orden de milisegundos y recolectarlos en una base de datos para análisis posterior.

La herramienta está compuesta por los siguientes elementos principales (Ver **Figura 2-17**)

- *Resource mScopeMonitor*: captura el estado de los recursos del sistema.
- *Event mScopeMonitor*: herramienta para generar trazas de log que permiten identificar los límites de ejecución de cada petición. Esta información permite inferir las dependencias de las peticiones y correlacionar eventos.
- *mScopeTransformer*: transforma los datos recolectados en un esquema estructurado.
- *mScopeDB*: base de datos principal donde centraliza toda la información recolectada para análisis avanzados.

Para validar la herramienta los autores aplicaron un *benchmark* a un sistema de servicios web, y verificaron la capacidad de registrar eventos y reconstruir la traza completa para cada transacción ejecutada.

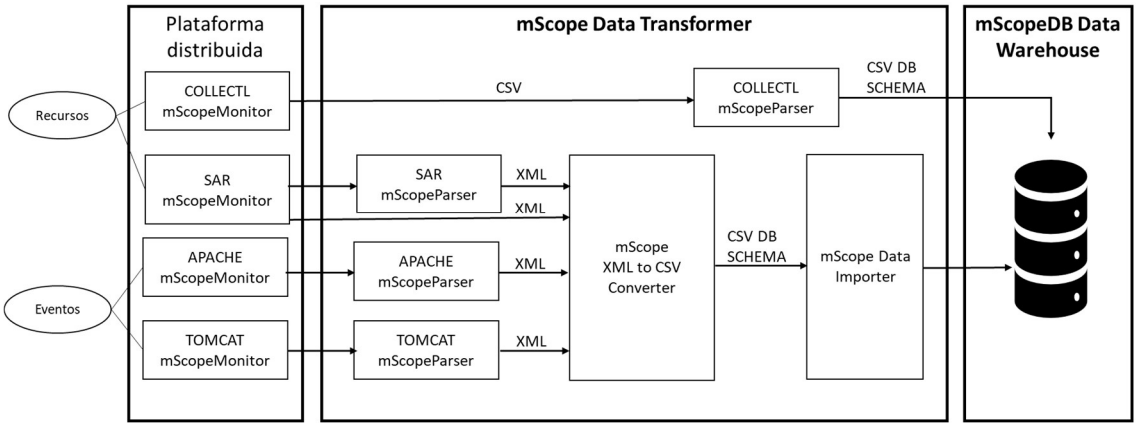


Figura 2-17: Elementos y flujo de datos en milliScope

2.10 Análisis y conclusiones

En la **Tabla 2-1** se muestra un resumen comparativo de las principales características de los trabajos mencionados anteriormente. El resumen expuesto evidencia que el tema del análisis del rendimiento en sistemas distribuidos lleva varios años de trabajo. Todos los trabajos previos están de acuerdo en que el problema de identificar fallas en sistemas distribuidos es complejo, en que los archivos log son una fuente viable, y a veces la única, para detectar el problema, pero la cantidad de datos y la complejidad de los sistemas en sí, hace inviable hacer esta tarea manualmente. También se puede inferir que el tema no ha sido abordado en Latinoamérica, probablemente porque los sistemas distribuidos son un tema relativamente nuevo en esta región.

En cuanto a la estrategia de recolección de datos se pueden identificar las siguientes estrategias:

- Utilización de APIs para generar logs especializados.
- Análisis de las peticiones de red.
- Análisis de los logs de la aplicación en un enfoque de caja negra.

La información utilizada para hacer el análisis del rendimiento se puede clasificar en las siguientes categorías:

- Flujos de operación, los cuales son inferidos automáticamente de la información recolectada.

- Eventos, que indican la ocurrencia de un error o cambio particular.
- Métricas de rendimiento: indican la carga del nodo: CPU, memoria, procesos, etc.

La medición de rendimiento principalmente se realiza midiendo el tiempo de latencia de las peticiones, en algunos casos solo se expresa el valor para que el usuario hiciera su análisis, en otros casos se compara con un valor de referencia para detectar la degradación del rendimiento. Dado que el rendimiento está directamente relacionado a flujos de operaciones, el método de visualización más común son los grafos.

Para poder dar solución al problema planteado se pueden utilizar varias de las estrategias encontradas en el estado del arte:

- Capacidad de inferir el flujo de operación global y de cada componente con un enfoque de caja negra, a partir de los archivos de log de la aplicación. Sin requerir APIs especiales o formatos especiales.
- Medición de los tiempos de latencia, y comparación con tiempos de referencia en estado normal.
- Captura de diferentes tipos de información:
    - Latencia de flujos de operación.
    - Ocurrencia de eventos.
    - Métricas de rendimiento de los nodos.
- Arquitectura distribuida y modular, para poder procesar en línea el rendimiento.

Dado que no se encontró en esta búsqueda bibliográfica trabajos que aplicaran algoritmos de aprendizaje computacional a la información capturada para medir y diagnosticar el rendimiento, se puede concluir que esta es una línea de investigación válida para el presente trabajo. La incorporación de este tipo de técnicas puede ofrecer información adicional que facilite el diagnóstico, tal como es la detección de anomalías, ya que la interpretación del tiempo de latencia puede tener diferentes interpretaciones dependiendo del contexto.

Tabla 2-1: Comparación metodologías encontradas en el estado del arte.

Trabajo **Año** **País** **Citas**	**Estrategia recolección y preprocesamiento**	**Medida del rendimiento**	**Detección de problemas**	**Visualización y reportes**	**Validación y resultados**	**Restricciones:** **Procesamiento en línea** **Transparente para aplicación**
Netlogger [6] • 2000 • EUA • 24	API integrado al sistema envía mensajes con formato de log a un servicio que recolecta toda la información	Mide el tiempo entre eventos	No identifica explícitamente un problema.	Gráfica con 3 componentes: Línea de vida Línea de carga Puntos de eventos	Verificación en aplicación Radiance	• Si • No
Depuración de rendimiento de cajas negras [9] • 2003 • EUA • 230	Se procesan los logs del sistema, y se infieren los flujos de operación. Alg. Anidado Alg. Convolución.	Mide el tiempo promedio de los flujos.	Acumula los valores totales en nodos superiores	Grafo con flujo, tiempo promedio y cuenta.	Verificación con trazas de log fuera de línea.	• No • Si
WAP5 [11] **2006** **EUA** **129**	Captura peticiones de red por proceso. Infiere flujos de mensajes	Mide tiempo de latencia de mensajes		Gráfico con flujo de peticiones, tiempo de latencia.	Verificación en dos sistemas distribuidos CoDeeN y Coral PlanetLab	• Si • Si Detecta flujos a nivel de peticiones de red
Detección de anomalías de ejecución en través de análisis de log [12] • 2009 • China • 102	Se generan log-keys a partir de los mensajes no estructurados.	Se crea Autómata de estado finito para modelar el flujo de ejecución. Se mide intervalo de tiempo de las transiciones de estados.	Modelo Gaussiano para comparar con valores típicos	No tiene visualización. Numero de transiciones que presentan anomalías Anomalías por número de ciclos	Verificación controlada en Hadoop y SILK	• No • Si
ASDF [14] • 2010 • EUA, Singapur • 1	Arquitectura escalable por medio de módulos. Módulos tipo Caja negra y blanca. Métricas de rendimiento del nodo		Módulos de análisis Media y varianza de ventana de tiempo K Vecinos (KNN)	Alarmas y gráficos (no definidos) Identifica nodo(s) con fallas	Verificacion en Hadoop utilizando benchmark GridMix	• Si • Si No tiene muchos módulos disponibles

Tabla 2-1 (CONTINUACIÓN): Comparación metodologías encontradas en el estado del arte.

Trabajo • **Año** • **País** • **Citas**	**Estrategia recolección y preprocesamiento**	**Medida del rendimiento**	**Detección de problemas**	**Visualización y reportes**	**Validación**	**Restricciones:** • **Procesamiento en línea** • **Requiere intervenir aplicación**
Magnifier [15] • 2011 • China • 15	Se deben agregar trazas de log en puntos específicos del sistema		Comparación por medio de media y varianza. Componentes principales para detectar nivel de influencia	No especificado	Pruebas controladas en Alibaba	• Si • No Requiere modificar el sistema.
Distalyzer [5] • 2012 • EUA • 110	Compara características estadísticas de archivos log buenos con malos	Se tienen predefinidos log de buen y mal rendimiento	Pruebas t-test de Welch, Kolmogorov-Smirnov para identificar diferencias	Gráfica con red de dependencias que enfoca el problema	Se valio con log de tres aplicaciones: TriSort, Hbase y Transmission	• No • Si
Splunk [16] • 2014 • EUA • 19	Soporta muchas fuentes de datos. Los cuales son indexados con una interfaz gráfica muy amigable.	Usuario debe definir usando queries	Datos indexados pueden ser filtrados, transformados, sumarizados	Interfaz WEB amigable. Cuenta de eventos se muestra en histograma. Alarmas		• Si • Si Herramienta comercial. Se requiere conocer el sistema
MilliScope [19] • 2017 • EUA • 2	Tiene componentes especializados para capturar recursos y eventos del orden de milisegundos			Datos se recolectan en una DB para análisis posterior	Se valida detección de eventos en ambiente simulado	• No • No Requiere componentes especiales.

3. Marco teórico

En esta sección se describen los conceptos más relevantes que están relacionados con el desarrollo de la presente propuesta.

3.1 Sistemas distribuidos

Los sistemas distribuidos son aquellos que están conformados por múltiples nodos o componentes, cada uno de los cuales cumple una función determinada. Dichos componentes están normalmente en computadores distribuidos a través de una red y coordinan sus acciones a través del intercambio de mensajes [20]. Las principales ventajas de los sistemas distribuidos radican en su capacidad de procesamiento y su tolerancia a fallos. Dicho sistemas agrupan varios paradigmas de computación tales como *clusters*, supercomputación, computación en malla (*grid computing*), servicios en la nube (*cloud computing*) [21] y microservicios (*microservices*) [22][23].

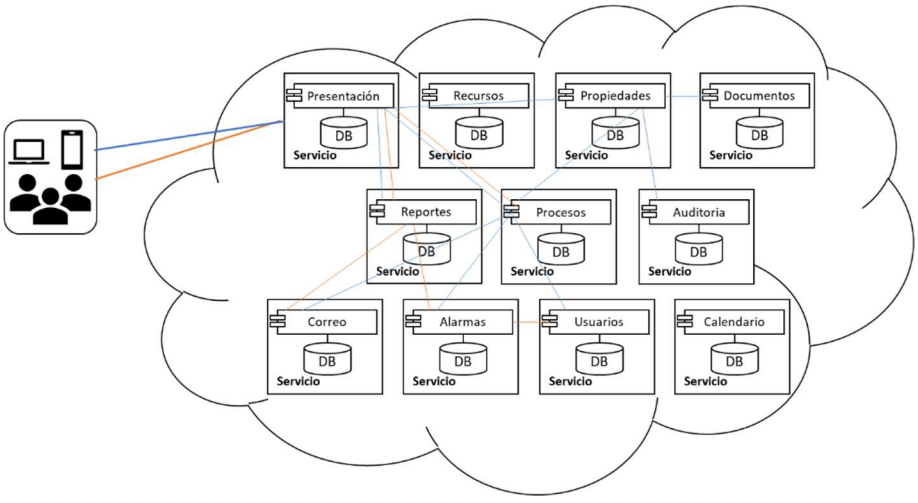


Figura 3-1: Sistema distribuido utilizando microservicios y sus interacciones [8]

En un sistema distribuido la ejecución de una operación implica un flujo de operaciones y mensajes entre múltiples nodos tal como se muestra en la **Figura 3-1**. El tiempo de respuesta total de dicha operación está directamente relacionado con los tiempos de respuesta de cada nodo implicado en la transacción. Por lo tanto, el rendimiento global en un sistema distribuido depende del rendimiento individual de cada nodo. En este sentido, un deterioro del rendimiento de un nodo afecta otros nodos y afecta el tiempo de respuesta global, estos fallos se conocen en algunos casos como fallos en cascada (*cascade failures*)[24].

La arquitectura por microservicios es un tipo de sistema distribuido, donde cada componente del sistema distribuido es un proceso independiente llamado microservicio. Un microservicio es un componente de software que se ejecuta autónomamente, conformando un componente que es independientemente reemplazable y actualizable, tiene las siguientes características:

- Pequeño, enfocado en hacer una sola cosa
- Proceso independiente
- Comunicación a través de API's agnósticas
- Altamente desacoplado
- Capacidad de escalamiento horizontal.
- Permite configuraciones de alta disponibilidad, con esquemas de redundancia y balanceo de carga.
- Facilita la integración con otras tecnologías

3.2 Archivos de log

Los archivos de log son ficheros de texto plano generados automáticamente por casi todos los sistemas computacionales, que registran la actividad en tiempo real de una aplicación, de la red o del sistema operativo. Los registros de un archivo de log por lo general consisten en líneas con información semi-estructurada, que incluye por lo menos una estampa de tiempo, una categoría del registro de log tal como INFO, WARN, ERROR y un texto en lenguaje natural [25][26]. En la Figura 3-2 se muestra un ejemplo de un archivo de log de una aplicación desarrollada en lenguaje JAVA, donde se puede observar una estampa de tiempo, un texto que indica el hilo de ejecución, la clase donde se genera el log y por último

un texto en lenguaje natural definido por el desarrollador. Los archivos log contienen información que se puede relacionar con los problemas de rendimiento, sin embargo, es una tarea complicada ya que el formato no es estándar y el volumen de datos puede llegar a ser muy alto.

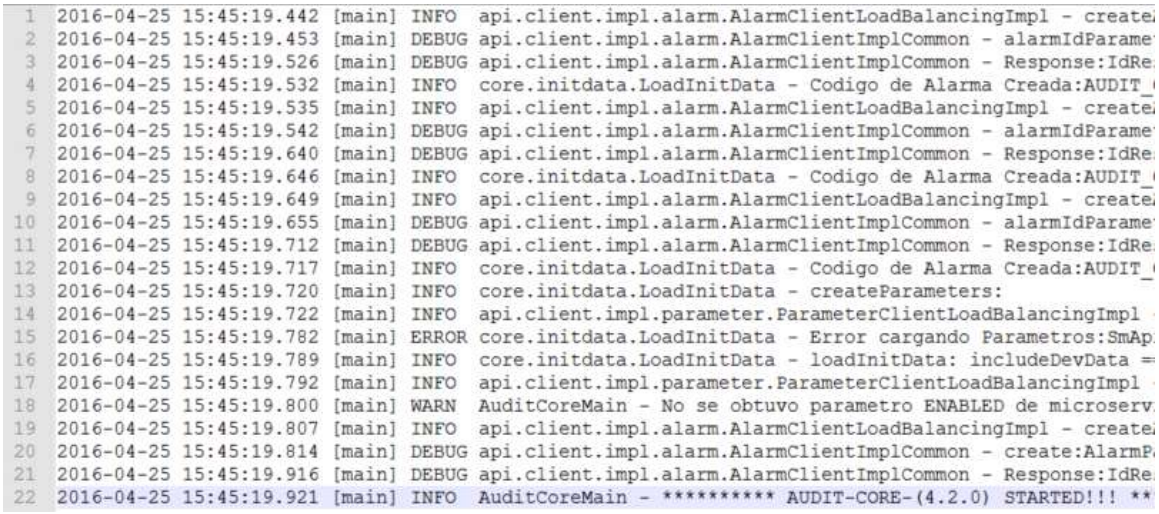


Figura 3-2: Archivo de log de aplicación JAVA.

El análisis de archivos de log se ha utilizado durante varias décadas para verificar el estado y comportamiento de un sistema en ejecución. En general se pueden categorizar las aplicaciones del análisis de log en los siguientes aspectos [27]:

- Depuración
- Rendimiento
- Seguridad
- Predicción
- Perfilamiento y Reportes

Los archivos de log nacieron de la necesidad de los programadores de poder inspeccionar el estado de un sistema y su comportamiento en ambiente de producción; de esta forma, los programadores pueden tener información para analizar la causa de una falla. El análisis de rendimiento permite a los administradores de los sistemas determinar el comportamiento del sistema desde el punto de vista de la carga del sistema y los tiempos de respuesta. La seguridad es un tema que también se ha abordado con el análisis de archivos de log, permitiendo detectar ataques, intrusiones y fraudes. Los archivos de log

también han ofrecido información que permite hacer predicciones, por ejemplo, en análisis de marketing. Por último, también se ha utilizado para perfilamiento de aplicaciones, ya que los archivos permiten identificar patrones de uso, utilización de recursos y comportamiento de los usuarios.

El procesamiento de archivos de log presenta varios retos a la hora de realizar análisis sobre estos:

- Falta de estandarización: Estos archivos no tienen establecido ningún tipo de estándar. Existen diferentes librerías que permiten gestionar la generación de log, pero cada una puede tener formatos diferentes. Además, normalmente hay un texto que corresponde a un mensaje en el lenguaje natural del programador.
- Gran Volumen de Información: El incremento de la capacidad de los sistemas actuales, con arquitecturas distribuidas, ha llevado a que la cantidad de información generada por los archivos de log sea cada vez más grande y por consiguiente más complicada de procesar. Esto se convierte en un problema aún más complejo si el objetivo es procesar y analizar toda esta información para generar resultados en tiempo real.

3.3 Aprendizaje de máquinas

El aprendizaje de máquinas (*Machine learning*) hace referencia al conjunto de técnicas y metodologías que permiten extraer conocimiento de los datos. Es un campo de investigación donde se cruzan la inteligencia artificial, la estadística y las ciencias computacionales. También es conocido como analítica predictiva o aprendizaje estadístico [28].

De una manera general las técnicas de aprendizaje de máquinas se pueden categorizar en tres grandes grupos como se muestra en la Figura 3-4.

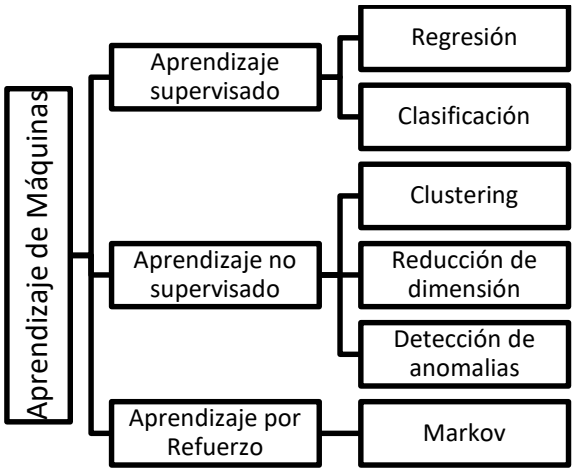


Figura 3-3: Categorías de técnicas de aprendizaje de máquina[29]

Las técnicas de aprendizaje supervisado (Supervised learning) se caracterizan por tener un conjunto de datos de entrada con su respectiva salida o respuesta. El objetivo de estos algoritmos es aprender los patrones de los datos y construir un modelo que permita predecir el valor de la salida ante nuevos datos de entrada. Dentro de esta categoría hay dos tipos de algoritmos:

- Regresión: La salida es un número continuo.
- Clasificación: la salida es una categoría o clase, donde se tienen dos o más posibles valores de respuesta.

La construcción de modelos de aprendizaje supervisado se realiza en tres fases:

1. Entrenamiento: Se le deben ingresar un conjunto de datos históricos de entrada con su respectiva salida. El algoritmo aprenderá los patrones presentes en los datos y creará un modelo.
2. Validación: Se verifica con un conjunto de datos históricos, diferente al usado en el entrenamiento, la diferencia entre los resultados históricos y los calculados por el modelo.
3. Predicción: Se aplica el modelo a nuevos datos para obtener un valor esperado.

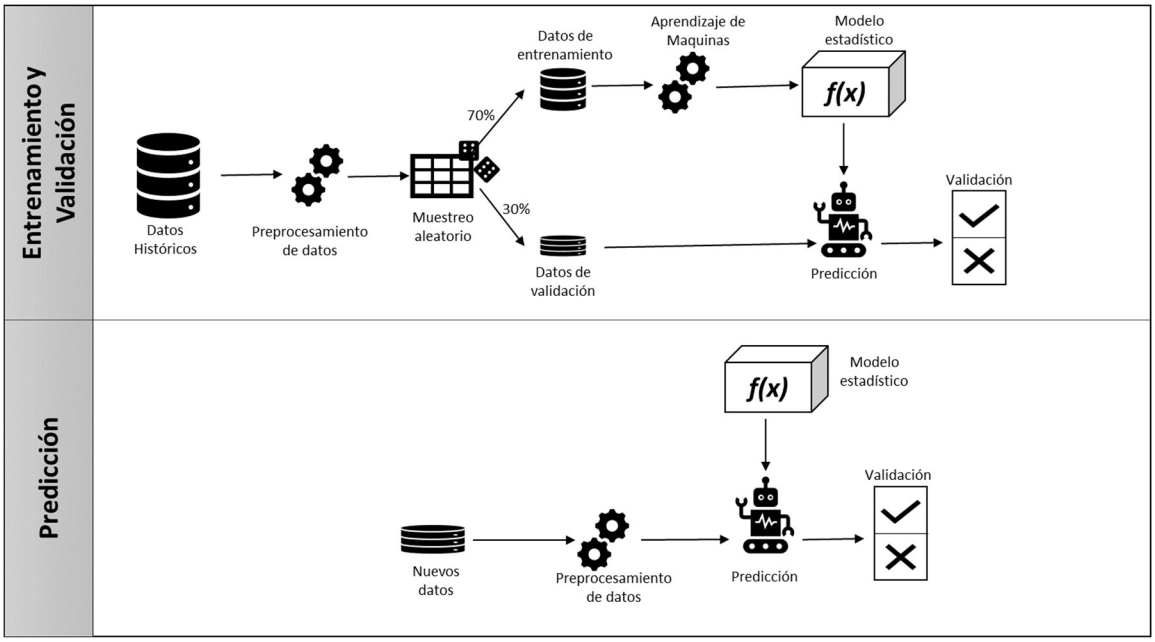


Figura 3-4: Proceso aprendizaje supervisado [29]

En el aprendizaje supervisado, se busca hacer un modelo de los datos existentes, y con este, hacer predicciones precisas sobre datos nuevos (con las mismas características de los que se analizaron inicialmente).

Se habla de generalización cuando un modelo puede hacer predicciones precisas sobre los datos nuevos. Es decir, el modelo es capaz de generalizar los datos de entrenamiento a los datos de prueba.

Dependiendo de la cantidad de datos disponibles y el procesamiento que se haga de estos, la generalización en algunas ocasiones puede ser inexacta o incompleta (infiere parte, pero no todos los patrones desconocidos)[30].

El concepto de *Overfitting* hace referencia a los casos en los que se construye un modelo estrechamente ligado a las particularidades de los datos de entrenamiento y se obtiene un modelo que funciona bien en los datos de entrenamiento, pero que no puede generalizarse a nuevos datos. El concepto de *underfitting* hace referencia a cuando se escoge un modelo demasiado simple, que no brinda una discriminación de los datos adecuada y por lo tanto, hace malas predicciones. Se considera que el underfitting es un problema menos frecuente que el overfitting [31]. La **Figura 3-5** muestra cómo se verían los diferentes fitting en un

ejemplo práctico. La elección del grado polinomial del orden correcto es muy importante para evitar los problemas de overfitting o underfitting en la regresión[29].

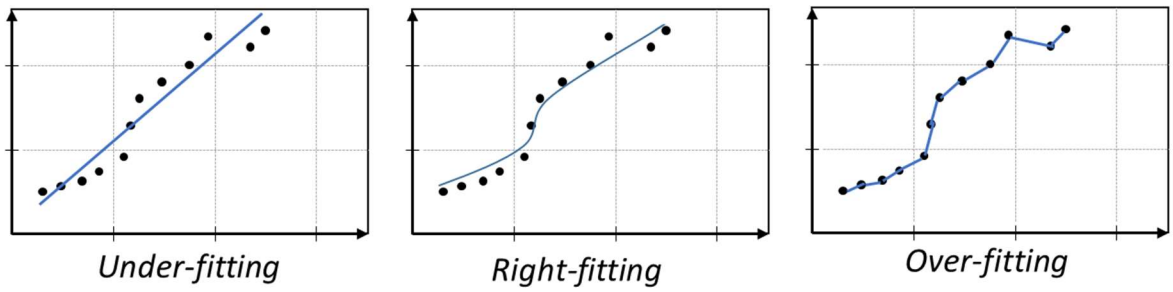


Figura 3-5: Under-fiiting vs Over fitting [29].

Las técnicas de aprendizaje no supervisado (*unsupervised learning*) se utilizan cuando en los datos históricos no se tiene la clase o categoría de salida. En estos casos el objetivo es aprender patrones de los datos que permitan obtener una mejor comprensión de estos e identificar patrones similares que permitan agrupar los datos en determinadas clases. En esta categoría se tienen tres tipos de algoritmos:

- Aglomerados (Clustering): identifica grupos de datos que tienen algún grado de similitud.
- Reducción de dimensión: el objetivo es simplificar un conjunto de datos con gran cantidad de entradas, de modo que el número de variables sea menor. Las variables seleccionadas deben ser las más influyentes en los datos.
- Detección de anomalías: también conocido en inglés como *outlier detection* es la identificación de eventos u observaciones los cuales no cumplen con el patrón esperado según los datos históricos.

El proceso en el aprendizaje no supervisado es similar al supervisado, sin embargo, la validación no se puede se hacer con los datos históricos, sino que es más una validación a nivel de conocimiento del negocio o problema.

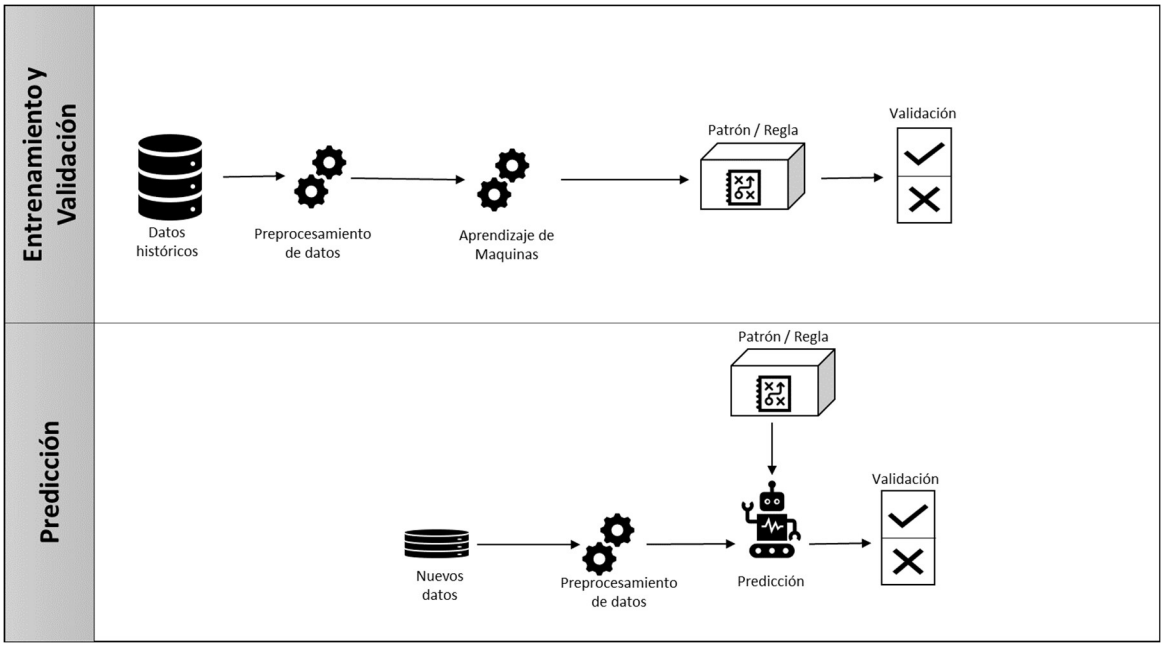


Figura 3-6: Proceso de aprendizaje no-supervisado [29]

3.3.1 Normalización de datos

La normalización de datos busca que la magnitud de los valores de las diferentes variables sea similar. Cuando hay variables que tienen valores mucho más grandes que otros, estos valores pueden afectar el algoritmo de aprendizaje, afectando la validez de las predicciones. Uno de los métodos más comunes es el escalado Min-Max. Este método utiliza los valores mínimos y máximos para convertir todos los valores en un rango de 0 a 1.

$$X_{normalized} = \frac{(X - X_{min})}{(X_{max} - X_{min})}$$

Otra técnica es la estandarización, la cual utiliza la media y la desviación estándar para convertir a valores con media 0 y desviación estándar 1.

$$Z = \frac{(X - \mu)}{\sigma}$$

3.3.2OneClass SVM

El algoritmo *OneClass SVM* (OCSVM) es un tipo particular de las técnicas de máquinas de soporte vectorial. Este tipo de SVM es un caso especial de la clasificación biclase en las que el algoritmo de clasificación se entrena con un solo tipo de clase para la posterior detección de patrones atípicos [32]. De manera similar a las máquinas de soporte el algoritmo calcula la frontera de decisión basada en hiperplanos denominados vectores de soporte. Si los datos de entrada se encuentran en el área de vectores típicos son etiquetados con el valor 1. En caso contrario, son etiquetados con el valor -1 y son considerados valores atípicos u *outliers*.

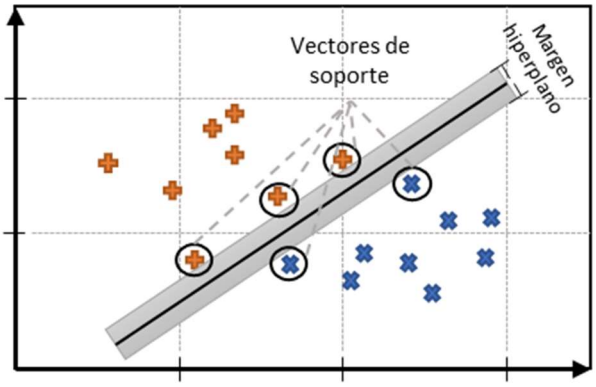


Figura 3-7: Hiperplanos de las máquinas de soporte vectorial SVM

Las máquinas de soporte SVM permite utilizar diferentes funciones *kernel*. Las más comunes son los tipos lineal, polinómico, sigmoide y gaussiano con base radial o (RBF). El *kernel* seleccionado en este trabajo es el RBF ya que es el más utilizado para el algoritmo OCSVM [32]. Adicionalmente el algoritmo utiliza dos parámetros de ajuste: “nu” y “gamma”. El parámetro “nu” tiene un valor entre 0 y 1, y define que tan ajustada se encuentra la frontera de decisión con respecto a los vectores de soporte. El parámetro “gamma” determina que tan ancha es la campana gaussiana en el kernel RBF, por lo que el óptimo ajuste de este parámetro determina la eficacia del algoritmo.

Para realizar la validación de los resultados, una de las formas más completas de representar el resultado de la clasificación binaria es usar la matriz de confusión (*confusion matrix*), la cual permite visualizar fácilmente la relación entre la clasificación calculada y la real [33]. La **Figura 3-8** muestra un modelo de matriz de confusión.

Clasificación Correcta	Clasificado como:	
	+	-
+	Verdaderos Positivos	Falsos Negativos
-	Falsos Positivos	Verdaderos Negativos

Figura 3-8: Matriz de confusión[33].

3.3.3Regresión multivariada

En la regresión simple se tiene una variable independiente para una variable dependiente, sin embargo, cuando se analizan casos de uso reales, generalmente se tiene más de una variable independiente y son estos casos los que se conocen como regresión multivariada [29]. En este tipo de análisis la ecuación se estructura de la siguiente forma:

$$y = m_1x_1 + m_2x_2 + m_3x_3 + \ldots + m_nx_n$$

Donde cada una de las variables independientes se representa con x, siendo m sus coeficientes correspondientes.

Las métricas más utilizadas para evaluar el desempeño de un modelo lineal son el error cuadrático medio o RMSE (*Root Mean Squared Error*), y el coeficiente de determinación o R^2 (*R-Squared*).

El error cuadrático medio se refiere a la raíz cuadrada de la media de los errores al cuadrado. Este valor indica que tan cerca están los valores predichos de los valores reales,

por lo que un valor bajo significa que el desempeño del modelo fue bueno. La unidad del RMSE va a ser la misma que la de la variable objetivo. La ecuación correspondiente al RMSE es:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i-\hat{y}_i\right)^2}$$

El coeficiente de determinación o R^2 es la proporción de varianza total en la variable dependiente explicada por la variable independiente [29]. Es un valor entre 0 y 1, entre más se acerque a 1, indica un mejor ajuste del modelo.

$$R^2 = \frac{\sum SSR}{\sum SST}$$

4. Definición del modelo

En este capítulo se explica el modelo propuesto para detectar los problemas de rendimiento. El modelo propuesto aborda tres aspectos principales tal como se muestra en la **Figura 4-1**.

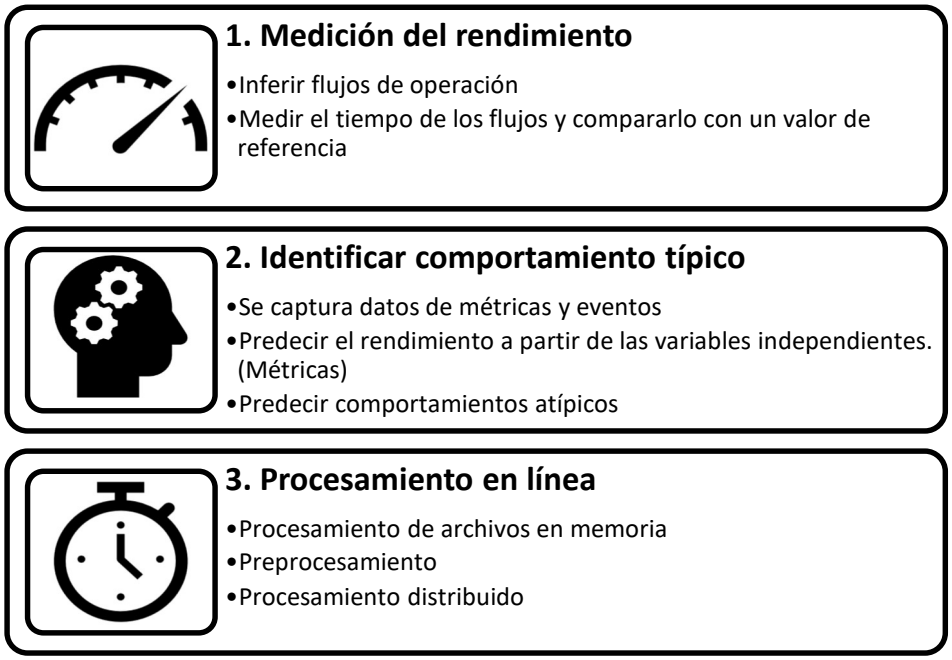


Figura 4-1: Aspectos incluidos en el modelo propuesto.

1. **Medición del rendimiento**: a partir de los archivos log se infieren los flujos de operación del sistema distribuido y se miden los tiempos de duración de cada flujo. Inicialmente se debe medir los tiempos de duración del sistema en un estado considerado normal con el fin de establecer tiempos de referencia. El rendimiento se mide con un indicador resultante de la relación entre las medidas de tiempo en cada instante y el tiempo de referencia.

2. **Identificar comportamiento típico**: se utilizan técnicas de inteligencia computacional para aprender el comportamiento típico del sistema.
3. **Procesamiento en línea**: el diseño de la herramienta permite procesar toda la información de los archivos log y de las métricas en línea. Para lograrlo se utiliza procesamiento en memoria y distribuido.

4.1 Modelo del rendimiento de un sistema

Antes de definir cómo se va a medir el rendimiento de un sistema distribuido se debe definir un modelo de medición. El rendimiento es una medida que está directamente relacionada con el tiempo de respuesta de un sistema. Sin embargo, esto no quiere decir que un tiempo de respuesta de 10 segundos sea peor que un tiempo de 500 milisegundos, pues esto depende de la aplicación que se esté ejecutando. Es muy diferente consultar un registro en una base de datos a procesar un archivo de texto de 100GB. Lo anterior indica que el rendimiento debe ser una medida que relaciona el tiempo de respuesta en un momento dado contra el tiempo de respuesta normal o esperado para una funcionalidad específica.

El tiempo normal de respuesta de un sistema distribuido es difícil de determinar, ya que hay muchos factores involucrados: capacidad del hardware, velocidad de la red, asignación de recursos del sistema operativo, componentes del sistema, datos manejados, etc. Sin embargo, si se mide en diferentes oportunidades el intervalo de tiempo entre el lanzamiento de una petición y su respuesta, se podría inferir desde esta información el tiempo típico de respuesta. Los sistemas distribuidos son multiusuario y multitarea, por lo que diferentes usuarios van a intentar lanzar la misma petición y cada usuario espera que el sistema responda como si fuera el único. Un sistema de calidad debe ser capaz de mantener el tiempo de respuesta en el valor esperado, o por lo menos con una variación aceptable para los usuarios. A medida que el tiempo de respuesta está más alejado del tiempo normal significa que el rendimiento del sistema está disminuyendo.

El umbral que determina cuando se considera un tiempo aceptable o cuando es una falla lo determina arbitrariamente el diseñador del sistema.

Considerando todo lo anterior, se podría modelar un sistema que tiene como entradas una secuencia de instrucciones o peticiones, y cuya salida sería el tiempo de ejecución de

dichas instrucciones. En condiciones ideales este tiempo estaría definido únicamente por las instrucciones como tal y por unos factores controlados como las características del hardware y de la red. Sin embargo, existen otros factores no controlados que incrementan el tiempo de respuesta del sistema: carga de los recursos de hardware, congestión de la red, cantidad de hilos y procesos simultáneos, errores de ejecución y fallos de hardware. Todo lo anterior se puede observar gráficamente en la Figura 4-2.

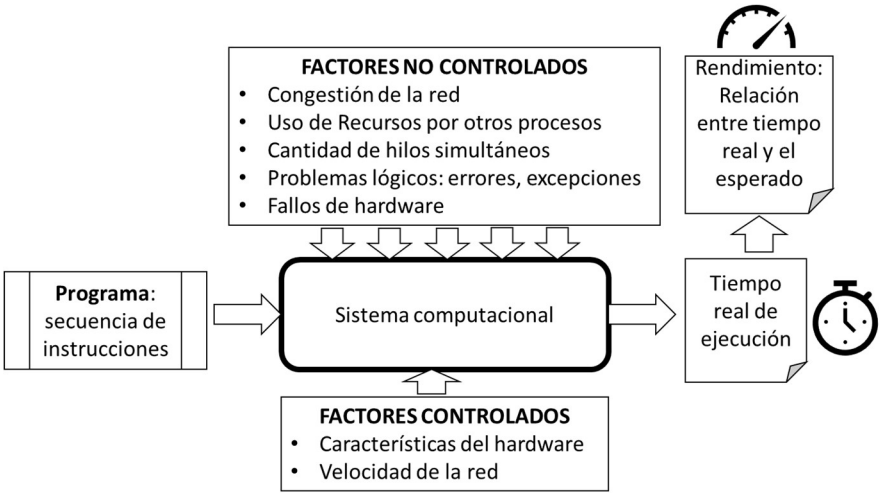


Figura 4-2: Modelo de rendimiento para un sistema computacional

En un sistema distribuido, este modelo se vuelve más complejo, ya que los factores no controlados se duplican por cada nuevo nodo en el sistema, incrementando la información que se debe procesar para analizar una falla de rendimiento. Para facilitar el análisis, la medición del rendimiento se realiza de manera independiente por cada componente o proceso ejecutable del sistema distribuido, tal como se muestra en la **Figura 4-3**. En este gráfico se muestra que el componente de la capa de presentación tiene un indicador de rendimiento del 55%, este a su vez tiene interacciones con otros tres componentes, de los cuales solo uno presenta un rendimiento bajo. Esto podría indicar que el bajo rendimiento de la capa de presentación se deba en buena parte al componente que presenta un 50% de rendimiento. Sin embargo, si seguimos viendo el flujo de llamados, vemos que este componente llama a otro que tiene un indicador de rendimiento del 12%. Probablemente este último sea la causa raíz del problema de rendimiento del sistema distribuido. Para corregir el problema, se debe verificar el funcionamiento de dicho componente, analizar si

se trata de un problema del software en sí, o si se trata de un problema generado por factores externos como la carga de la CPU o la cantidad de memoria utilizada.

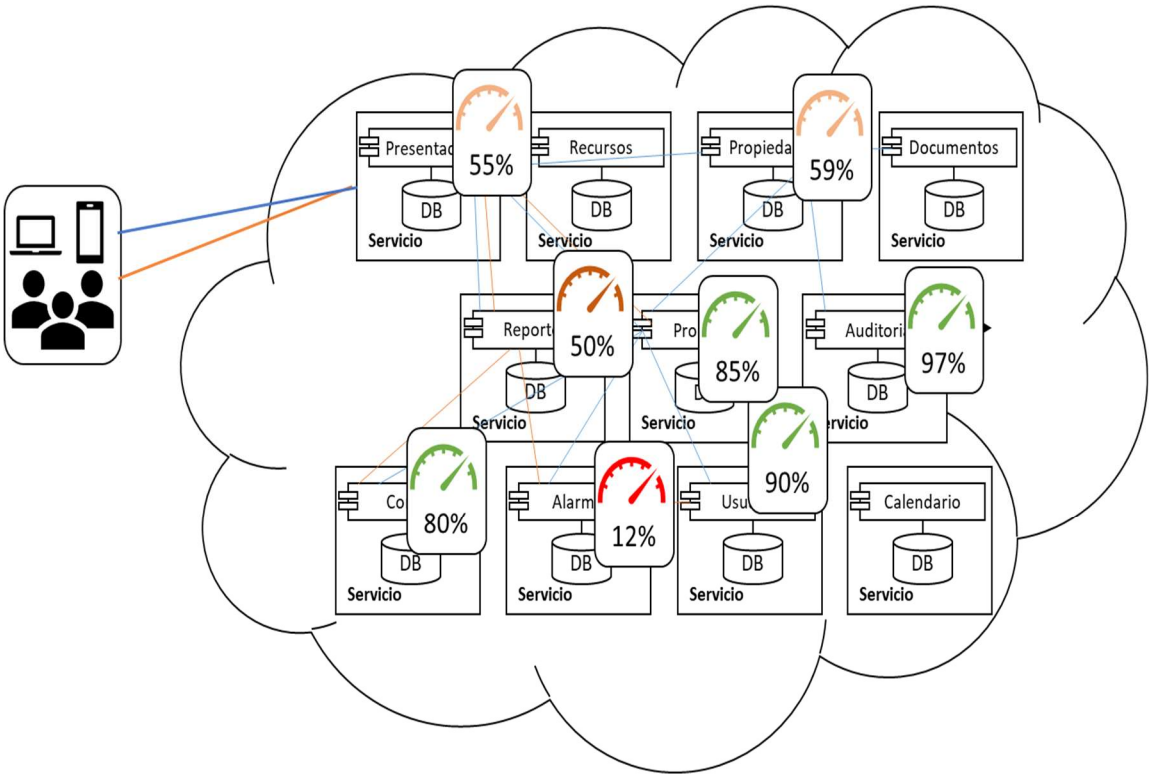


Figura 4-3: Sistema distribuido con indicadores de rendimiento por cada componente.

4.2 Definición de términos

A continuación, se definen algunos términos que se deben tener claros para comprender los capítulos siguientes.

Tabla 4-1: Definición de términos.

Término	Descripción
Nodo o host	Cada servidor del sistema distribuido. Las variables no controladas son independientes entre los diferentes hosts.
Componente (component)	Cada elemento ejecutable del sistema distribuido. Cada componente es una fuente de información de log.
Hilo de ejecución (thread)	Los sistemas multitarea operan con hilos de ejecución que corren paralelamente. Las instrucciones en cada hilo se ejecutan secuencialmente.
Traza (trace)	Una traza es un mensaje que permite identificar una ubicación en el código fuente del sistema. Una secuencia de mensajes de traza permite inferir un flujo de ejecución y determinar la duración de una traza a otra.
Evento (event)	Un evento corresponde a la ocurrencia de un suceso. Un evento puede ser un mensaje de error o de advertencia. También pueden utilizarse para número de procesos o número de usuarios. Las variables tipo evento son contadores enteros, por lo tanto, su rango es [0, INF).
LogKey	Se refiere a un tipo de mensaje de log, que puede ser tipo traza o tipo evento.
LogRecord	Registro de log, cada vez que aparece un logKey. Normalmente corresponde a cada línea del archivo de log. Debe cumplir la siguiente estructura: <Tiempo> <Thread> <logKey> [<Información Extra>]
logPath	Indica una relación secuencial entre dos logKey tipo traza. Tienen la siguiente estructura: LogKey Origen -> LogKey Destino
RemoteCall	Llamado remoto entre componentes. Tiene la estructura: LogKey Componente Origen -> LogKey Componente Destino

4.3 Medición del rendimiento

La medición del rendimiento busca obtener un indicador numérico para cada componente de un sistema distribuido. El enfoque propuesto es de tipo caja negra, es decir, no se requiere conocer, ni modificar la estructura o el funcionamiento del sistema a medir. La presente propuesta utiliza la información de los archivos de log para inferir los flujos de operación y medir su duración para calcular el rendimiento. El proceso se puede dividir en las siguientes etapas:

1. Captura de datos: creación de *threads*, *logKeys* y *logRecords*
2. Mapeo: Inferir flujos de operación: se crean los *logPaths*
3. Agrupamiento de las medidas en ventanas de tiempo
4. Cálculo valor de referencia
5. Cálculo de indicador de rendimiento

4.3.1 Captura de datos

La captura de datos busca procesar los archivos log para transformarlos en un esquema estructurado que permita inferir los flujos de operación de un componente. En la **Figura 4-4** se puede observar un ejemplo de cómo las líneas de log que aparecen en el archivo reflejan una secuencia de ejecución en el código fuente. En este ejemplo, que tiene dos clases, se puede observar que el flujo normal se refleja en tres líneas del archivo log. La ejecución inicia por el método *createEvaluationCode* de la clase *TransactionController*, desde la cual se llama al método *createEvaluacionCode* de la clase TransactionCoreBusinessImpl; dentro de este método se llaman otros denominados *validate*() y *save*(), si no se presenta ninguna excepción se registra en el log un mensaje "*createEvaluationCode ok*". También se puede observar que en los archivos log la secuencia de las líneas no necesariamente corresponde a la secuencia de los flujos de operación, esto se debe a que al ser un componente multitarea se manejan simultáneamente varios hilos de ejecución.

En un programa de software simple las líneas de código se ejecutan de manera secuencial. Si el componente es multitarea, por ejemplo, un servidor WEB, cada vez que se quiere ejecutar la secuencia de instrucciones se genera un nuevo hilo de ejecución. El sistema operativo es el encargado de controlar la ejecución de los diferentes hilos en paralelo. En

el archivo de log este paralelismo se evidencia al mezclarse mensajes de diferentes hilos. Para poder capturar la secuencia de operación desde el archivo log se requiere que el mensaje de log tenga una llave que permita diferenciar los diferentes hilos de ejecución.

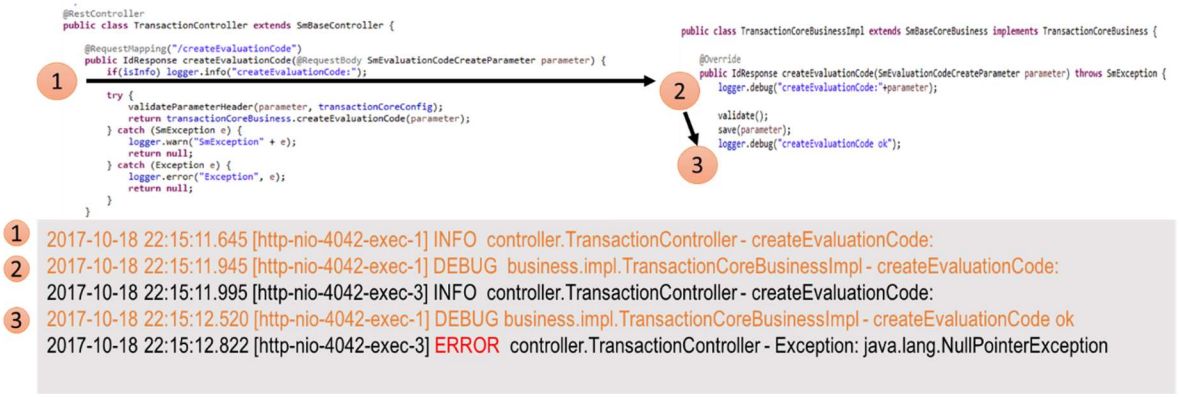


Figura 4-4: Código fuente y logs con información de traza

En el ejemplo de la **Figura 4-4** los registros de log tienen la siguiente estructura:

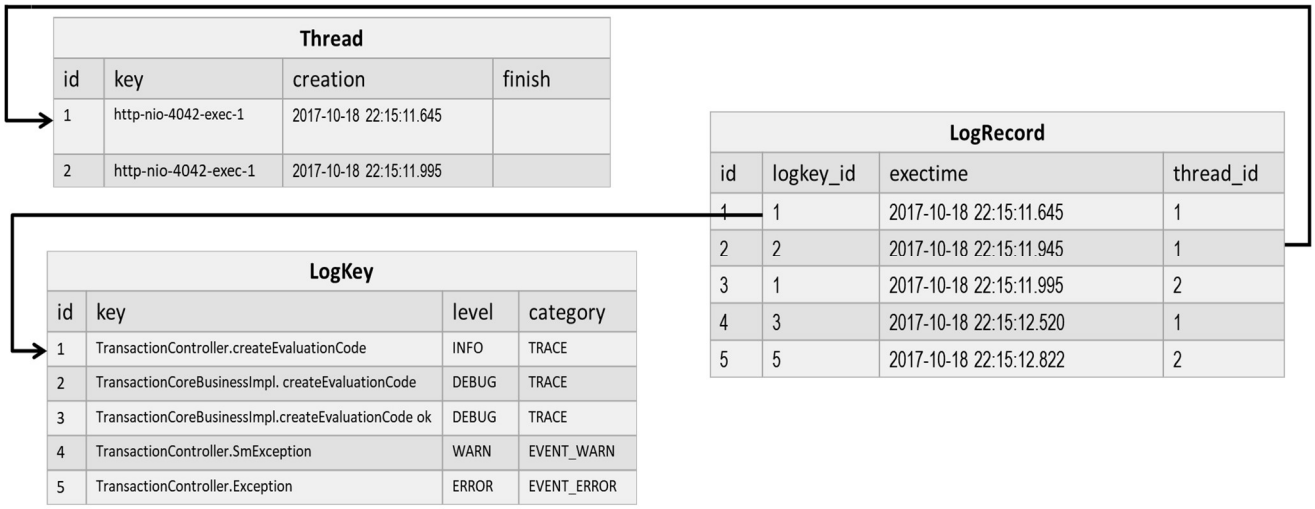
- **Estampa de tiempo**: 2017-10-18 22:15:11.645
- **Hilo de ejecución**: [http-nio-4042-exec-1]
- **Nivel o categoría del log**: INFO
- **Clase**: *controller*.*TransactionController*
- **Texto**: *createEvaluationCode*:

Como mínimo los registros de log deben tener tres tipos de datos:

- Estampa de tiempo: necesaria para poder medir el rendimiento.
- Hilo de ejecución: para poder inferir flujos de operación.
- Ubicación: código que permitan identificar un registro de log con una posición en un código fuente. Ejemplos: texto, clase, método, línea.

Si los mensajes de log cumplen con estas condiciones se puede crear un esquema estructurado donde se manejan tres tablas (ver **Figura 4-5**):

- *Thread*: registra cada hilo de ejecución.
- *LogKey*: corresponde a cada tipo o clase de mensaje de log.
- *LogRecord*: tabla principal donde se registra cada mensaje de log en el tiempo.



Thread

id	key	creation	finish
1	http-nio-4042-exec-1	2017-10-18 22:15:11.645	
2	http-nio-4042-exec-1	2017-10-18 22:15:11.995	

LogRecord

id	logkey_id	exectime	thread_id
1	1	2017-10-18 22:15:11.645	1
2	2	2017-10-18 22:15:11.945	1
3	1	2017-10-18 22:15:11.995	2
4	3	2017-10-18 22:15:12.520	1
5	5	2017-10-18 22:15:12.822	2

LogKey

id	key	level	category
1	TransactionController.createEvaluationCode	INFO	TRACE
2	TransactionCoreBusinessImpl. createEvaluationCode	DEBUG	TRACE
3	TransactionCoreBusinessImpl.createEvaluationCode ok	DEBUG	TRACE
4	TransactionController.SmException	WARN	EVENT_WARN
5	TransactionController.Exception	ERROR	EVENT_ERROR

Figura 4-5: Esquema de datos para información de archivos log.

Como se puede ver en el ejemplo, los registros de la tabla *LogKey* tienen una categoría. Esta categoría indica si se trata de un mensaje de traza o de evento. En el ejemplo, los eventos corresponden a excepciones que quiebran el flujo de ejecución. Para el caso de los eventos el hilo de ejecución no es relevante.

4.3.2 Parser

El *parser* es el artefacto encargado de leer la información cruda de los archivos log y convertirla a un formato estructurado para la herramienta. La solución debe permitir agregar diferentes tipos de *parser*, que se ajusten al formato de cada archivo de log particular. La salida del análisis de cada línea de archivo log será una estructura de datos con los campos mostrados en la **Tabla 4-2**. Estos campos correspondes a la información de un *logRecord*.

Tabla 4-2: Atributos de un logRecord.

Atributo	Descripción	Tipo
Timeexec	Estampa de tiempo	Tiempo
Host	Identificación del host	Identifica componente del sistema distribuido
Component	Identificación del componente	
Instance	Identificación de la instancia del componente	
Classname	Clase donde se crea el mensaje de log	Identifica logKey en el software. Información para categorizar
Method	Clase donde se crea el mensaje de log	
lineNumber	Clase donde se crea el mensaje de log	
logLevel	Nivel de prioridad del mensaje LOG	
Text	Texto sin formato definido	
Key	Identificador único del LogKey. Es creado a partir de los datos anteriores.	
eventCategory	Categoría del evento. Debe ser definido por el parser	Categoría del logKey
threadKey	Identificador del hilo	Hilo
RemoteCallKey	Identificador del llamado remoto	Llamados remotos
userKey	Identificador del usuario	Atributos relevantes adicionales
tenantKey	Identificador del dominio. Aplica solo para sistemas multitenant	

4.3.3 Inferencia de flujos de operación

Con la información de traza capturada se pueden inferir los flujos de operación del sistema distribuido a nivel del componente, a este proceso se le denomina mapeo. El proceso general se describe como se muestra en la **Figura 4-6**. Se seleccionan los registros de log correspondientes a un hilo determinado y se ordenan cronológicamente. Dado que es un flujo secuencial se crean los conectores o *logPath* correspondientes. La diferencia de las estampas de tiempo es el tiempo de ejecución entre el *LogKey* inicial y el *LogKey* destino.

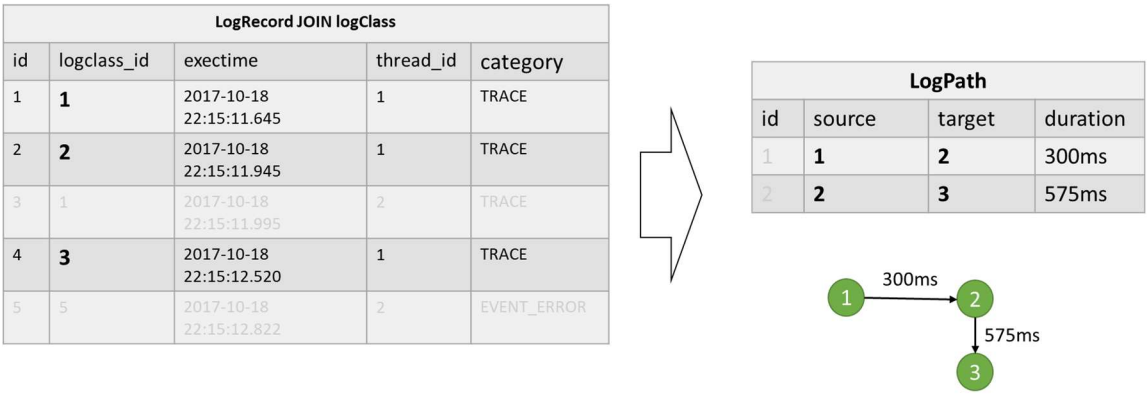


Figura 4-6: Inferencia de flujos de operación.

El proceso anterior crea los flujos que se ejecutan dentro de cada componente, pero no permiten identificar los llamados entre componentes. En la sección 4.3.8 se explica el proceso para hacer el mapeo de los llamados remotos.

4.3.4 Agrupamiento de las medidas en ventana de tiempo

En la tabla *logRecords* de la **Figura 4-6** se registran cada uno de los registros del archivo log, con su estampa de tiempo. Cuando ya se conocen los flujos de operación, entonces se puede calcular la duración de cada *logPath* por medio de la diferencia entre las dos estampas de tiempo respectivas. Para limitar la cantidad de datos a procesar, se define una ventana de tiempo de un minuto, en la cual las medidas de duración de los *logPaths* son agregadas. Como resultado de este agrupamiento se crean un registro denominado *pathMeasure* el cual tiene los siguientes atributos:

- Fecha y hora del registro.
- *logPath*.
- Duración promedio en la ventana de tiempo.
- Duración máxima en la ventana de tiempo.
- Desviación estándar de la duración en la ventana de tiempo.
- Número de registros capturados.

De esta manera, aunque se ejecuten miles de *logPaths* en un minuto, estos serán reemplazados por un solo registro de tipo *pathMeasure*. Se escogió un minuto como ventana de tiempo, por considerarse un periodo suficiente para limitar los datos, pero no tan largo como para afectar una respuesta en línea.

4.3.5 Cálculo del valor de referencia

Después de tener definidos las conexiones entre los LogKey, se procede a calcular el valor de referencia para la medida de rendimiento para cada logPath. Se escoge un rango de tiempo donde el comportamiento del sistema distribuido se pueda catalogar como normal y se calcula, para cada logPath, la duración promedio, la desviación estándar máxima y el valor máximo. Antes de calcular y almacenar estos valores se hace un procesamiento previo para eliminar casos atípicos. Para esto se calcula la media y la desviación estándar de los datos preseleccionados y se eliminan de la muestra los datos cuyo valor no estén en el rango $[\mu - 3\sigma, \mu + 3\sigma]$, que por definición de la distribución normal son el 99.6% de los datos (Ver **Figura 4-7**)

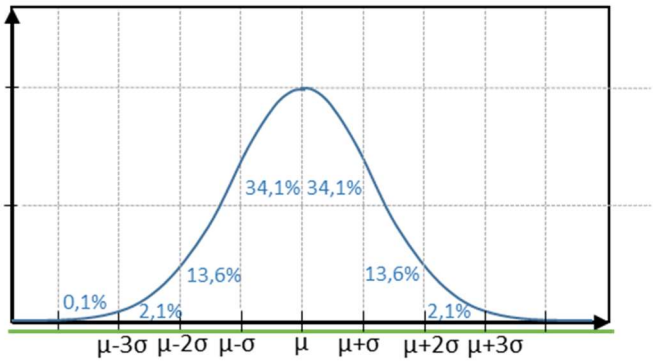


Figura 4-7. Distribución normal. Rango donde se encuentra el 99.6% de la población

4.3.6 Cálculo del indicador del rendimiento

Para calcular el rendimiento de cada logPath se utiliza la función mostrada en la **Figura 4-8**. Este indicador fue definido empíricamente y utiliza como parámetro la duración promedio y la desviación estándar de referencia. Este método es una función lineal que está limitada al rango [0, 1]. Donde 1 significa que el rendimiento es adecuado.

```
Def perfomance(avgref, stdref, avg):
    #avgref : duración promedio de referencia
    #stdref : desviación estándar de referencia
    #avg    : duración promedio actual

    if stdref < 0.01: stdref = 0.01
    f = (1-((avg - avgref) / (stdref*2)))*0.9
    if f > 1: f=1
    if f < 0: f=0
    return f
```

Figura 4-8: Función para calcular rendimiento.

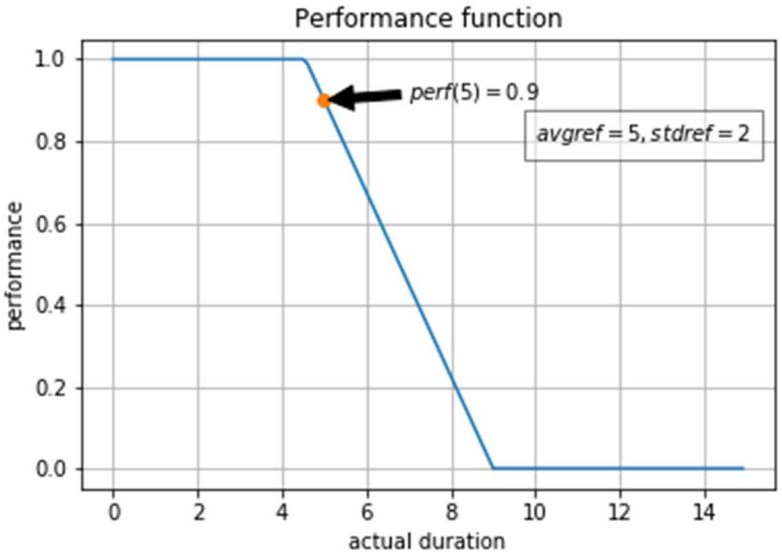
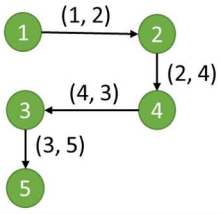


Figura 4-9: Gráfica función de rendimiento.

La **Figura 4-9** muestra el comportamiento de la función de rendimiento para una duración promedio de referencia de 5 y una desviación estándar de 2. Si la duración actual son 5 segundos, igual al promedio, el rendimiento es de 0.9, lo que se puede interpretar que el rendimiento es bueno, pero no excelente. Si la duración disminuye se va acercando a 1, que indicaría un buen rendimiento, porque está por debajo del promedio. Cuando la duración es mayor a 9, que corresponde a la media más dos veces la desviación estándar (5+2*2) entonces se considera que el rendimiento está totalmente degradado y se le asigna el valor 0.

Después de calcular el indicador de rendimiento por cada LogPath, se procede a calcular el indicador de rendimiento global del componente. Debido a que los flujos no se ejecutan de modo uniforme para calcular el rendimiento total del componente se utiliza un promedio ponderado por el número de ocurrencias de cada logPath tal como se muestra en la **Figura 4-10**.



logPath	Ref	Real	Count	Rendimiento (ref/real)
(1,2)	0,3	0,3	5	1,0
(2,4)	0,7	0,5	5	1,0
(4,3)	0,3	0,6	2	0,5
(3,5)	0,5	0,6	10	0,8

Rendimiento total componente:
(1*5+1*5+0,5*2+0,8*10)/22 = **0,86**

Figura 4-10: Cálculo del rendimiento del componente.

4.3.7 Cálculo del indicador de rendimiento mínimo

El rendimiento en un sistema multitarea, como son los sistemas distribuidos, es complejo de analizar porque simultáneamente se ejecutan múltiples hilos, los cuales pueden tener características de rendimiento diferentes. En la **Figura 4-11** se observa una gráfica de una prueba de carga realizada con la herramienta Jmeter a una plataforma WEB, el eje Y es el tiempo de respuesta en milisegundos y el eje X es el tiempo. Cada línea corresponde al tiempo de respuesta de una URL determinada. Se observa que a partir de las 15:15 el tiempo de respuesta aumento considerablemente para todas las URL, pero especialmente una se degrada totalmente. El indicador de rendimiento debe decir en este caso, que hay una disminución del rendimiento, pero, sin embargo, en su gran mayoría el sistema sigue operando. Para poder reflejar la presencia de casos extremos de mal rendimiento, como el mal de la figura, se define un nuevo indicador llamado rendimiento mínimo que refleje los peores tiempos de ejecución, sin importar si su cantidad es mucho menor. La función

para este nuevo indicador es similar a la función de rendimiento, diferenciándose en que en vez de utilizar la duración promedio utiliza el valor máximo. Ver **Figura 4-12**

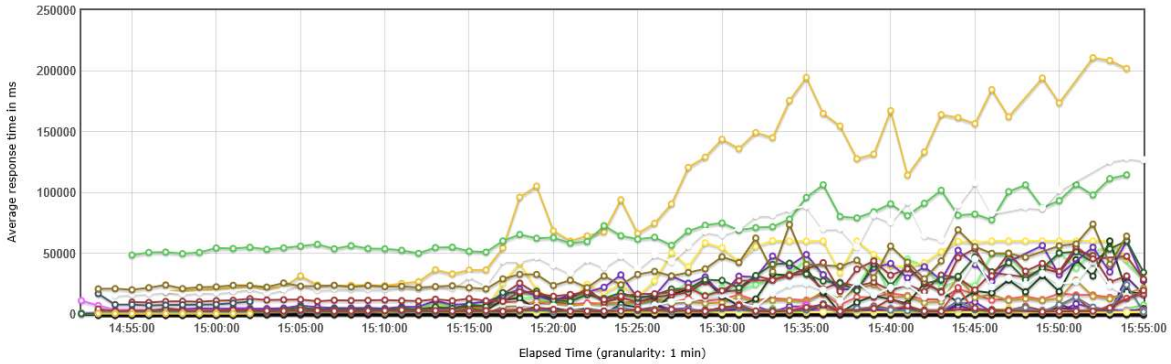


Figura 4-11: Medida de tiempo de respuesta de un sistema WEB.

```
Def perfomanceMin(avgref, stdref, vmax):
    #avgref : duración promedio de referencia
    #stdref : desviación estándar de referencia
    #vmax   : duración maxima actual

    if stdref < 0.01: stdref = 0.01
    f = (1-((vmax - avgref) / (stdref*2)))*0.9
    if f > 1: f=1
    if f < 0: f=0
    return f
```

Figura 4-12: Función para calcular rendimiento mínimo.

4.3.8 Mapeo de llamados remotos

La información del log corresponde a un componente individual, por lo tanto, no se puede obtener el flujo de llamados entre componentes del sistema distribuido. Para poderlos inferir, se asume que los mensajes de log correspondientes a peticiones entre componentes incluyen un identificador que permite relacionar mensajes entre diferentes componentes. Para inferir los llamados entre componentes se utiliza los datos de log capturados en la tabla *logRecords* y el atributo *RemoteCallKey* mencionado en la **Tabla 4-2**. El proceso para mapear los llamados remotos consiste en los siguientes pasos:

1. Recolectar todos los *logRecords* que tengan el atributo *RemoteCallKey* en una tabla única.
2. Agrupar los *logRecords* por *RemoteCallKey*.
3. Por cada grupo se ordenan los *logRecord* en orden cronológico.
4. Se recorre la secuencia de *logRecords*, cada vez que se detecte un cambio de componente se crea un llamado remoto.

4.4 Identificar comportamiento típico

En esta sección se explica cómo utilizar técnicas de aprendizaje de máquinas para obtener información adicional que ayude a identificar anomalías en las variables dependientes y, adicionalmente, para determinar cuándo una medida de rendimiento es normal de acuerdo con las métricas o factores no controlados del sistema. El proceso general de la presente propuesta está compuesto por cinco pasos principales tal como se muestra en la **Figura 4-13**.

1. Captura: se debe capturar la mayor cantidad de datos que ofrezcan información sobre todos los factores que puedan afectar el rendimiento.
2. Preprocesamiento: los datos son agrupados por ventanas de tiempo.
3. Creación de características: se crean nuevos datos a partir de los datos medidos. En este paso se crean las métricas y eventos calculadas a partir de la información capturada desde los archivos log.
4. Entrenamiento: en esta fase se aplican técnicas de aprendizaje de máquinas para aprender cuál es el comportamiento normal del sistema que permitan realizar la detección de anomalías y la predicción del indicador de rendimiento. Se hace la selección del conjunto de muestras de entrenamiento y se realiza la transformación de los datos.
5. Validación: se valida el modelo entrenado con un conjunto de datos de prueba.

Estos pasos se desarrollan independientemente para cada uno de los componentes del sistema distribuido.

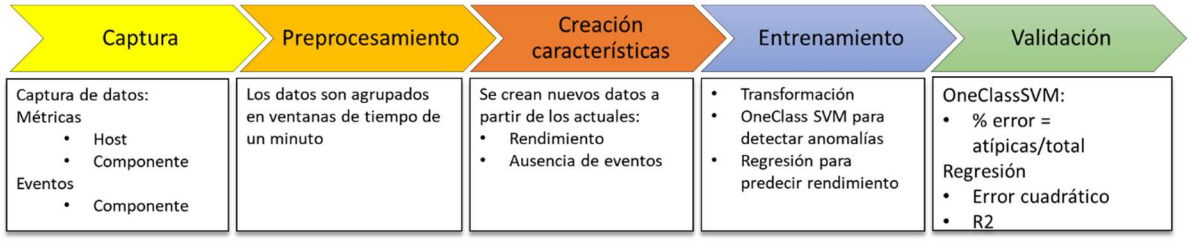


Figura 4-13: Proceso aprendizaje de máquinas.

4.4.1 Captura de datos

La información dada por los archivos log permite inferir los flujos de operación, medir el rendimiento y detectar eventos del componente tales como errores y advertencias. Sin embargo, como se mostró en el modelo de rendimiento, este depende de muchos otros factores. Debido a eso, para asegurar datos que permitan tener información suficiente para los algoritmos de aprendizaje de máquinas, se debe complementar los datos con otras fuentes de información. A continuación, se explican los tipos de datos que se van a capturar.

La información capturada por el sistema se categoriza en dos tipos principales:

- Métricas: variables numéricas que indican un nivel de alguna variable del sistema, por ejemplo, el porcentaje de CPU y memoria utilizado.
- Eventos: son variables de enteros positivos que indican el número de veces que ocurre un suceso dado, por ejemplo, el contador de errores de red en un periodo, o la cantidad de mensajes de error que aparece en el archivo log.

Tanto las métricas como los eventos pueden venir de diferentes fuentes de información. Los archivos log son la primera fuente de datos, donde los mensajes de error y advertencia son categorizados como eventos. Para poder analizar el rendimiento una fuente de información obligatoria es el sistema operativo de cada nodo, ya que de éste se puede obtener el estado de carga del host, por medio de métricas como el estado de la CPU, memoria RAM, utilización de disco y utilización de red.

Eventos capturados desde archivos log

Los mensajes de log que corresponden a registros tipo ERROR, WARNING y CRITICAL indican la ocurrencia de eventos excepcionales que rompen el flujo normal del sistema. Registrar este tipo de eventos, presenta el inconveniente de que la cantidad de tipos de mensaje es indefinida y puede llegar a ser muy alta. Para solventar esto, se define un conjunto de categorías de eventos para los archivos log (ver **Tabla 4-3**), las cuales permiten agrupar los tipos de mensajes en grupos relacionados al tipo de problema expresado. Dicha categorización se realiza en la etapa de captura de datos, y las reglas de categorización deben ser definidas por una persona que conozca el sistema monitoreado y el tipo de mensajes que genera. Inicialmente, los eventos se asocian a los tipos más generales: WARNING, ERROR Y CRITICAL. Contar con eventos de todas las categorías permitirá al sistema ofrecer una mejor información de diagnóstico.

Tabla 4-3: Categorías de eventos capturados desde archivos log.

Tipo evento	Descripción
COMM_ERROR	Error de comunicación
AUTH_ERROR	Error de autenticación
DB_ERROR	Error de base de datos
VIEW_ERROR	Error de capa de presentación
DATA_ERROR	Error de datos
WARNING	Advertencia
ERROR	Error
CRITICAL	Error critico
EVENT_BOOT	Evento de inicio del componente
TRACE_BOOT	Trazas del inicio del componente

Datos capturados desde el sistema operativo

Las métricas y eventos del host deben ser capturados por un proceso especializado para tal tarea. Las métricas de carga del host indican el estado del hardware y del sistema operativo en un momento dado, lo cual afecta directamente el rendimiento. Dichas medidas son entregadas por el sistema operativo utilizando comandos o funciones ofrecidos para ello. En la **Tabla 4-4** se muestran las variables capturadas para el nodo.

Tabla 4-4: Métricas y eventos de host.

Tipo evento	Descripción
CPU total	% de uso de CPU total
CPU usuario	% de uso de CPU por parte del usuario
CPU idle	% disponible de CPU
Memoria	% de uso de la memoria
SWAP	% de uso de la memoria SWAP o virtual
Usuarios	Número de usuarios autenticados en el sistema operativo
PIDs	Número de los procesos activos en el sistema operativo
CNXs	Número de conexiones de red
Uso disco	% de uso del disco duro
Rata IO disco	Rata de IO de disco duro
Rata IO red	Rata de IO de interfaces de red
Rata error red	Rata de errores reportados por interfaces de red
Rata drop red	Rata de paquetes eliminados por interfaces de red
Archivos abiertos	Número de archivos abiertos por el sistema operativo

Otras fuentes de datos

Además de los archivos log y del sistema operativo, se pueden adicionar componentes de monitoreo especializados para obtener más datos que complementen la información antes mencionada. Para el presente trabajo se desarrollaron los siguientes componentes de monitoreo adicionales:

1. Monitor de microservicio JAVA: utilizando un servicio web tipo REST se consultan métricas del estado del componente.
2. Monitor de PostgreSQL: se conecta a una base de datos PostgreSQL y obtiene métricas sobre el estado de las conexiones y consultas.
3. Monitor de Apache Tomcat: se conecta al servicio de gestión y consulta el estado del servidor.

En las siguiente tres tablas se especifican los datos capturados por cada uno de estos componentes de monitoreo.

Tabla 4-5: Métricas y eventos de monitor para microservicios JAVA.

Variable	Descripción
MAX(memused)	Memoria utilizada por la JVM
MAX(heapused)	Memoria tipo HEAP utilizada por la JVM
MAX(nonheapused)|	Memoria tipo NONHEAP utilizada por la JVM
MAX(threads)	Número de hilos activos
MAX(classes)	Número de clases cargadas
MAX(sessions)	Número de sesiones activas

Tabla 4-6: Métricas y eventos de monitor para PostgreSQL.

Variable	Descripción
MAX(conns)	Conexiones activas a la base de datos
MAX(locks)	Numero de queries con bloqueo exclusivo

Tabla 4-7: Métricas y eventos de monitor para Apache Tomcat.

Variable	Descripción
MAX(memused)	Memoria utilizada por la JVM
MAX(threads)	Número de hilos activos
MAX(threadsBusy)	Número de hilos ocupados
MAX(workers)	Número de procesos controladores

4.4.2 Preprocesamiento de datos

La etapa de preprocesamiento tiene como objetivo principal resumir todos los datos capturados durante una ventana de tiempo definida. De este modo la cantidad de datos que se deben transmitir al componente centralizado estará controlado y no será un factor importante de sobrecarga para el sistema distribuido.

La ventana de tiempo debe ser un tiempo suficiente para no aumentar el tráfico en la red, pero tampoco debe ser tan amplia que los resultados del análisis demoren tanto que ya no sean útiles. Teniendo en cuenta lo anterior, se selecciona una ventana de tiempo de un minuto, para la cual los datos son agrupados de la siguiente manera:

- Para las métricas de componentes de monitoreo se entrega el valor máximo de las muestras tomadas.
- Para todos los eventos se entrega el valor acumulado durante la ventana de tiempo.

4.4.3 Creación de características

Esta fase busca crear nuevas variables a partir de los datos existentes que sean útiles para el objetivo del problema. La característica más relevante es el cálculo de los indicadores de rendimiento, lo cual fue ampliamente explicado en el capítulo 4.3. Adicional a esto, se deben crear nuevos datos cuando hay ausencia de eventos. Es decir, cuando no sucede un evento determinado, no se registra nada en la base de datos de la herramienta ya que nada dispara el proceso. Debido a esto, para cada ventana de tiempo, se deben buscar los eventos de todas las variables y cuando no se encuentren crear el nuevo registro con valor cero.

A partir de los datos capturados de los archivos log se pueden crear nuevas variables relacionadas con el estado de carga del componente. La **Tabla 4-8** muestra las nuevas variables creadas a partir de los archivos log.

Tabla 4-8: Variables creadas a partir de datos de archivos log.

Tipo evento	Descripción
threadsCount	Número de hilos creados en la ventana de tiempo
logCount	Número *logRecords* capturados
logTraceCount	Número *logRecords* tipo traza capturados
logEventsCriticalCount	Número *logRecords* tipo evento critico capturados
logEventsErrorCount	Número *logRecords* tipo evento error capturados
logEventsWarningCount	Número *logRecords* tipo evento advertencia capturados

4.4.4 Entrenamiento del modelo

En este paso se hace el entrenamiento de los algoritmos de aprendizaje. En esta fase se cuenta con una matriz de datos con la información mostrada en la **Figura 4-14** por cada componente del sistema distribuido. La cantidad de variables puede variar por cada componente, ya que todas las fuentes de datos no aplican para todos los tipos de monitoreo, por ejemplo, el monitor de Apache Tomcat solo aplica para la interfaz WEB.

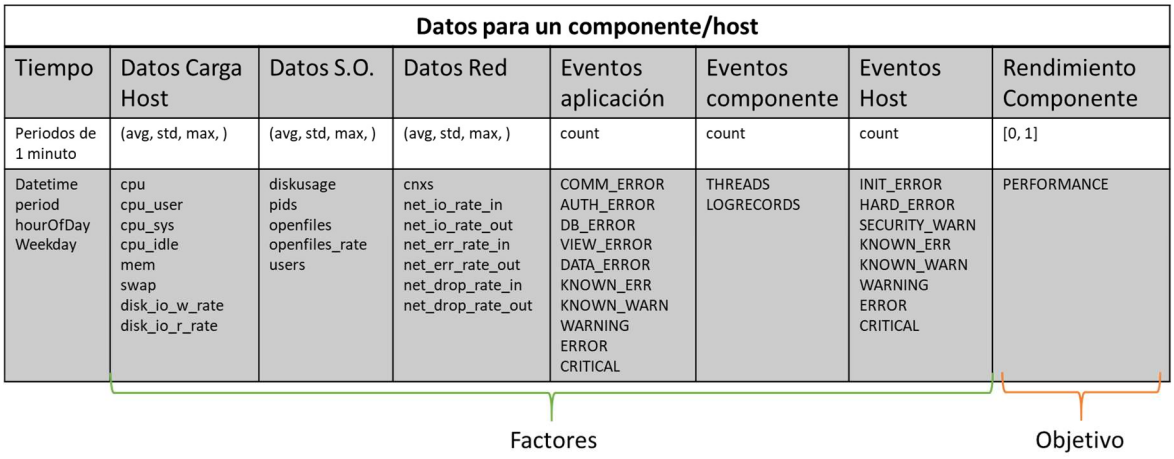


Figura 4-14: Datos para aprendizaje por cada componente.

El objetivo es utilizar los datos mencionados anteriormente para realizar dos tipos de predicción:

- **Predicción del rendimiento:** comparar el rendimiento esperado con el rendimiento real.
- **Predicción de anomalías:** detectar si las variables se están comportando de manera típica.

El entrenamiento de los algoritmos de aprendizaje está compuesto por varias actividades, las cuales deben realizarse de manera independiente para cada uno de los algoritmos:

1. Selección de variables
2. Limpieza de datos
3. Transformación de datos
4. Muestreo de datos de entrenamiento
5. Entrenamiento

6. Validación con datos de prueba

La limpieza de datos consiste en eliminar aquellos registros que no tengan todos los datos completos. Los datos capturados tienen diferentes escalas, lo que afecta la eficiencia del entrenamiento de los algoritmos de aprendizaje, por eso en la fase de transformación los datos son escalados con el método Min-Max explicado en el marco teórico. De este modo, todas las variables tienen un rango entre cero y uno.

Modelo para detección de anomalías

Para la predicción de anomalías se seleccionan todos los factores independientes como variables de entrada al modelo, y de esta manera identificar cuando estos factores se están comportante de una manera atípica. El algoritmo seleccionado para detectar anomalías es el OCSVM, el cual según trabajos previos presenta un buen desempeño para detectar comportamientos atípicos [34].

Modelo para predicción del indicador del rendimiento

Para realizar la predicción del rendimiento se utiliza la regresión lineal multivariada, donde se tiene como variable dependiente el indicador de rendimiento. Todas las demás variables son posibles variables independientes. Para determinar las variables que se consideran métricas que estén relacionadas con el rendimiento se hace un análisis exploratorio de todas las variables y un análisis de correlación. En la **Figura 4-15** se muestra un ejemplo del análisis exploratorio de las variables, donde se pueden observan cuatro gráficas:

- Un gráfico de dispersión del valor máximo de utilización de CPU vs la medida del rendimiento. En esta figura se observa que los datos están muy dispersos, lo cual significa que, aunque la CPU es un factor directamente relacionado con el rendimiento, los otros factores presentan una influencia considerable en el rendimiento.
- Un gráfico del valor máximo de la CPU contra el tiempo, para observar el comportamiento de la variable en el tiempo. Para este caso se observa que la variable se mueve entre el rango de 0 a 100.
- Gráfico de caja, donde se observa cómo es la distribución de los datos respecto a la media. Este gráfico muestra que la media es menor al 20%, y la mayoría de los datos se encuentran entre 0 y 30%, pero hay valores atípicos entre el 60% y 100%.

- Histograma con distribución de frecuencia. Se observa que la mayoría de las muestras presentan un valor entre 0 y 10% y van disminuyendo a medida que el valor es más alto.

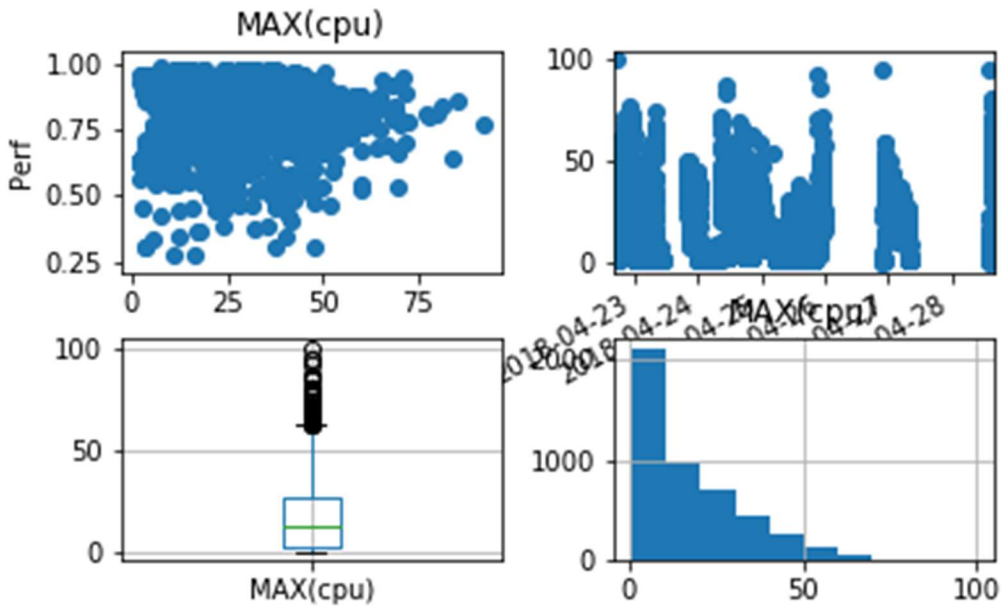
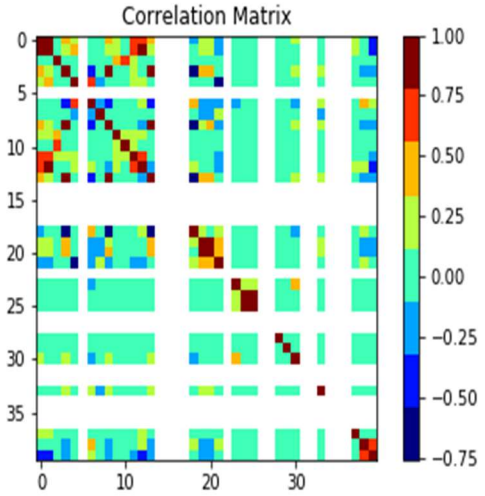


Figura 4-15: Ejemplo análisis exploratorio variable CPU.

En la **Figura 4-16** se muestra el resultado del análisis gráfico de la matriz de correlación, donde se puede visualizar por medio de una escala de colores el nivel de correlación entre variables. En conclusión, se puede ver que en general la correlación entre variables es baja, y las correlaciones que se encuentran eran las esperadas, tales como son la utilización de la CPU con el tráfico de la red. Dado que la correlación más relevante que se quiere analizar es la de todas las variables o factores independientes con la variable objetivo que es el rendimiento, se crea la **Tabla 4-9**, donde se observa el valor de correlación del indicador de rendimiento contra los factores independientes. Como se mencionó anteriormente, el análisis se hace independiente por cada componente, por lo que en la tabla se muestra el valor de correlación para tres componentes: *web*, *alarm*, *smsecurity*.



	var name
0	MAX(cpu)
1	max(cpu_user)
2	MAX(cpu_sys)
3	MIN(cpu_idle)
4	MAX(mem)
5	MAX(swap)
6	MAX(diskusage)
7	MAX(pids)
8	MAX(cnxs)
9	MAX(disk_io_rate_w)
10	MAX(disk_io_rate_r)
11	MAX(net_io_rate_r)
12	MAX(net_io_rate_s)
13	MAX(openfiles)
14	SUM(net_err_rate_in)
15	SUM(net_err_rate_out)
16	SUM(net_drop_rate_in)
17	SUM(net_drop_rate_out)
18	MAX(memused)
19	MAX(threads)
20	MAX(threadsBusy)
21	MAX(workers)
22	COUNT(fail)
23	threadsCount
24	logCount
25	logTraceCount
26	logEventsCriticalCount
27	logEventsErrorCount
28	logEventsWarningCount
29	TRACE_BOOT
30	ERROR
31	DATA_ERROR
32	VIEW_ERROR
33	DB_ERROR
34	AUTH_ERROR
35	NET_ERROR
36	WARNING
37	EVENT_BOOT
38	PERFORMANCE
39	PERFORMANCE_MIN

Figura 4-16: Análisis de correlación.

Tabla 4-9: Análisis de correlación de las variables con el indicador de rendimiento.

Fuente	Variable	web	alarm	smsecurity
Host	MAX(cpu)	0.004740	0.153983	0.122730
Host	max(cpu_user)	0.060098	0.157743	0.292555
Host	MAX(cpu_sys)	0.004205	-0.008957	-0.000804
Host	MIN(cpu_idle)	-0.161723	0.098839	-0.383600
Host	MAX(mem)	0.083873	-0.241844	-0.227903
Host	MAX(swap)	NaN	NaN	NaN
Host	MAX(diskusage)	0.365894	-0.260040	0.045423
Host	MAX(pids)	0.000885	0.008449	0.207286
Host	MAX(cnxs)	-0.155616	0.027802	-0.046727
Host	MAX(disk_io_rate_w)	0.020749	-0.004536	0.030001
Host	MAX(disk_io_rate_r)	-0.002987	-0.027398	0.006763
Host	MAX(net_io_rate_r)	0.051794	0.092070	0.152170
Host	MAX(net_io_rate_s)	0.169926	0.400732	0.260451
Host	MAX(openfiles)	-0.184286	0.071829	-0.049048
Host	SUM(net_err_rate_in)	NaN	NaN	NaN
Host	SUM(net_err_rate_out)	NaN	NaN	NaN
Host	SUM(net_drop_rate_in)	NaN	NaN	NaN
Host	SUM(net_drop_rate_out)	NaN	NaN	NaN
Microservicio	MAX(memused)		0.064529	0.013989
Microservicio	MAX(heapused)		0.027750	0.022518
Microservicio	MAX(nonheapused)		0.033782	-0.003866
Microservicio	MAX(threads)		0.015993	-0.053693
Microservicio	MAX(classes)		0.003235	0.043993
Microservicio	MAX(sessions)		0.145401	0.100195
Microservicio	COUNT(fail)		-0.003995	NaN
Postgres	MAX(conns)		0.011008	0.014602
Postgres	MAX(locks)		0.034745	0.046865
Log	threadsCount	-0.047689	-0.017988	-0.044025
Log	logCount	-0.000515	0.022672	-0.032291
Log	logTraceCount	0.000001	0.024148	-0.032255
Log	logEventsCriticalCount	NaN	NaN	NaN
Log	logEventsErrorCount	NaN	NaN	NaN
Log	logEventsWarningCount	-0.000385	-0.020365	0.005063
Log	TRACE_BOOT	-0.042807	0.135408	0.047257
Log	ERROR	-0.037868	0.158556	0.128399
Log	DATA_ERROR	NaN	NaN	NaN
Log	VIEW_ERROR	NaN	NaN	NaN
Log	DB_ERROR	-0.029939	-0.048470	-0.161755
Log	AUTH_ERROR	NaN	NaN	NaN
Log	NET_ERROR	NaN	NaN	NaN
Log	WARNING	NaN	NaN	NaN
Log	EVENT_BOOT	-0.032712	0.024054	0.014186
Tomcat	MAX(memused)	0.157699		
Tomcat	MAX(threads)	-0.214003		
Tomcat	MAX(threadsBusy)	-0.182009		
Tomcat	MAX(workers)	0.151966		
Tomcat	COUNT(fail)	NaN		

Como resultado del análisis exploratorio y de la correlación se descartan las variables resaltadas en la **Tabla 4-9** para el análisis de regresión. Las que tiene como valor NaN son así porque no presentan variación, entonces no hay una correlación. También se descartan las que se consideran eventos excepcionales, y que su ocurrencia no implica causalidad en la afectación del rendimiento. Todas las otras variables son seleccionadas como entradas al modelo de regresión.

4.4.5 Validación del modelo

Para entrenar y validar el modelo propuesto es necesario tener datos, para obtenerlos este trabajo incluye el desarrollo de una herramienta de software (ver capítulo 5) y la aplicación de ésta a un sistema distribuido desplegado en un ambiente de pruebas (ver capítulo 6). Para adquirir los datos para el entrenamiento y la validación se seleccionan datos del ambiente de pruebas en condiciones típicas de operación, que corresponden aproximadamente a tres días de monitoreo. De este conjunto de datos se selecciona de manera aleatoria un 70% de datos para entrenamiento y los datos restantes se reservan para la validación.

Validación del modelo para detección de anomalías

Para validar el modelo entrenado con la técnica OCSVM, se aplica el modelo entrenado al conjunto de datos de validación y se obtiene el porcentaje de error, que corresponde a las muestras marcadas como atípicas. Esta métrica es suficiente para medir qué tan bueno es el modelo ya que como en los datos solo se tiene un valor o clase de salida, no se puede tener falsos positivos.

$$\% \text{ error: } \frac{Número\ de\ muestras\ etiquetadas\ como\ atipicas}{Numero\ total\ de\ muestras\ de\ validación} * 100$$

Se seleccionó el *kernel* RBF, que es el más utilizado para detección anomalías [34]. Para obtener los parámetros de ajuste “nu” y “gamma” que ofrezcan un mejor desempeño, se ejecuta un proceso iterativo donde se prueban todas las combinaciones de dichos parámetros para seleccionar los valores que muestren un error menor. En la **Tabla 4-10**

se muestran los resultados de la validación de la detección de anomalías por cada componente. La tabla incluye el porcentaje de error, los parámetros finales utilizados, el número de muestras de entrenamiento y el de validación. Se observa que el error está alrededor del 1%, lo que para el propósito del presente trabajo se considera como un error no significativo.

Tabla 4-10: Validación detección de anomalías por componente usando OCSVM.

#	Componente	%Error	nu	gamma	n train	n test
0	microserviceregister	0,993	0,01	0,01	2719	1166
1	parameter	0,922	0,01	0,02	2712	1163
2	audit	0,960	0,01	0,02	2709	1161
3	alarm	0,996	0,01	0,04	2711	1163
4	device	0,962	0,01	0,03	2704	1160
5	smsecurity	0,850	0,01	0,01	2706	1161
6	user	0,850	0,01	0,01	2706	1161
7	transaction	1,108	0,01	0,01	2708	1161
8	devinterface-trx	1,108	0,01	0,01	2708	1161
9	concurrent	1,072	0,01	0,03	2704	1160
10	location	0,924	0,01	0,08	2707	1161
11	notification	0,997	0,01	0,07	2708	1161
12	smdemo-web	1,116	0,01	0,13	2688	1153
13	etl	1,072	0,01	0,01	2706	1161
14	directory	0,921	0,01	0,06	2606	1117
15	custommodelcore	0,924	0,01	0,01	2707	1161

Validación del modelo para predicción del indicador del rendimiento

En la **Tabla 4-11** se presentan los resultados de las métricas del modelo de regresión entrenado, que incluyen el error cuadrático medio (RMSE), el coeficiente de determinación (R^2) y el tamaño de los conjuntos de entrenamiento y validación. Teniendo en cuenta que el valor de la regresión es un valor entre cero y uno, también se muestra el porcentaje de error. En la tabla se puede observar que la cantidad de muestras del conjunto de entrenamiento y validación son diferentes para cada componente, esto es así ya que al ser un método de aprendizaje supervisado, requiere tener el dato de la variable objetivo (indicador de rendimiento), y este valor no está en todas las muestras ya que depende del nivel de carga de cada componente.

El porcentaje de error máximo encontrado es del 6.1% que se considera aceptable, ya que lo que se busca es obtener un valor estimado basado en las métricas del nodo y del componente. El coeficiente de determinación no está cerca al valor ideal para ningún componente, esto indica que la variabilidad del rendimiento no es reflejada por el modelo. En la **Figura 4-17** se puede observar que la línea de predicción trata de seguir a la línea de rendimiento, pero no puede seguir la alta variabilidad que presenta. A pesar de esto, y teniendo en cuenta que el porcentaje del error promedio fue menor al 6.1%, no se descarta el modelo, ya que, para el objetivo final de la solución, no reflejar esta variabilidad no es un factor relevante.

Tabla 4-11: Validación de la predicción del rendimiento por componente.

#	Componente	RMSE	%Error	R2	n train	n test
0	microserviceregister	0,001006	0,10	-0,018117	940	404
1	parameter	0,005977	0,60	0,093528	2654	1138
2	audit	0,017158	1,72	0,064879	830	357
3	alarm	0,008349	0,83	0,222487	2653	1138
4	device	0,012992	1,30	-0,400794	431	185
5	smsecurity	0,007043	0,70	0,230111	1183	507
6	user	0,009587	0,96	0,281156	1356	582
7	transaction	0,003882	0,39	0,474638	1340	575
8	devinterface-trx	0,020997	2,10	-0,059463	307	132
9	concurrent	0,019158	1,92	0,029506	305	131
10	location	0,061561	6,16	0,173201	665	285
11	notification	0,003396	0,34	0,270232	2654	1138
12	smdemo-web	0,009787	0,98	0,374046	1432	615
13	etl	0,01715	1,72	0,00315	662	285
14	directory	0,015729	1,57	0,307382	348	150
15	custommodelcore	0,008603	0,86	0,162732	266	115

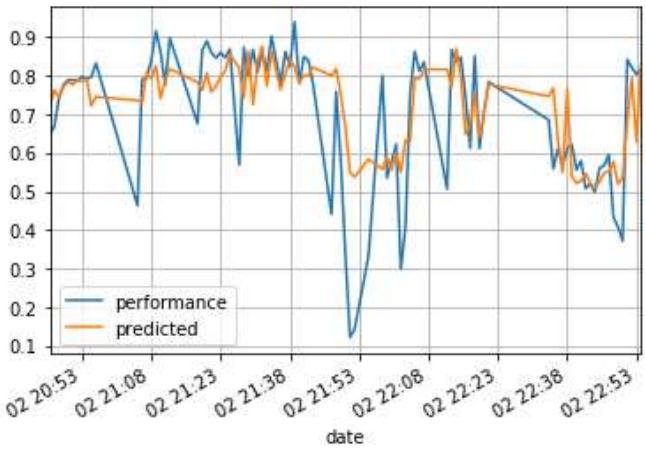


Figura 4-17: Rendimiento medido vs predicción.

4.5 Procesamiento en línea

El último aspecto que se debe tiene en cuenta en el presente trabajo es cómo realizar un procesamiento en línea, que ofrezca datos eficaces para diagnosticar problemas de rendimiento en un periodo de tiempo oportuno, que para esta implementación particular se define como límite de respuesta un lapso de cinco minutos. La **Figura 4-18** muestra el esquema general de la solución propuesta: agentes distribuidos en cada host capturan los datos de los archivos log, del host y de otros componentes de monitoreo. Los datos son preprocesados y enviados a un módulo maestro, que centraliza todos los datos y realiza la medición del rendimiento y el análisis predictivo. Toda esta información es entregada al usuario final a través de una interfaz de usuario, que muestra el estado de los diferentes componentes del sistema distribuido y gráficas con el comportamiento de las diferentes variables.

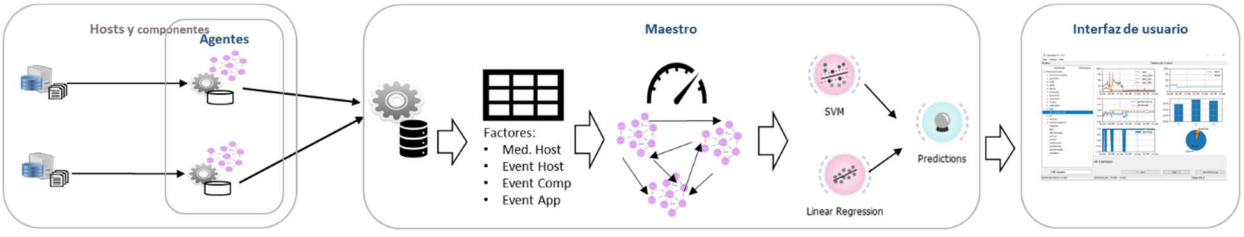


Figura 4-18: Esquema de procesamiento.

Para lograr un procesamiento en línea, se definen las siguientes tres estrategias:

- Captura de datos distribuida a través de los agentes.
- Preprocesamiento distribuido: los agentes agrupan los datos capturados en información correspondiente a ventanas de tiempo de un minuto.
- El procesamiento de datos se realiza, en la medida de lo posible, en memoria, para asegurar una mayor velocidad de procesamiento.

Estas estrategias están muy relacionadas con el diseño de la herramienta, el cual se va a explicar con más detalle en el capítulo 5.

5. Diseño de la herramienta LogMapper

El modelo expuesto en el capítulo anterior es implementado en una herramienta de software denominada LogMapper. El código fuente de esta herramienta se deja disponible como opensource y puede ser descargado desde Internet [35]. En este capítulo se explica a nivel general las funciones y características de cada componente de la solución.

La herramienta de software está compuesta por tres componentes:

- Agente (logmapper-agent)
- Maestro (logmapper-master)
- Interfaz de Usuario (logmapper-ui)

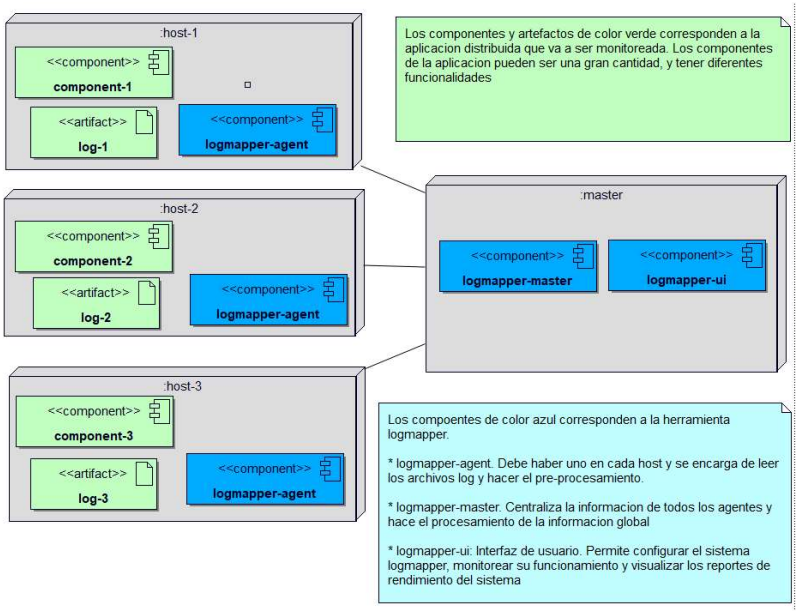


Figura 5-1: Arquitectura general LogMapper.

La herramienta está desarrollada en lenguaje Python 3.6. Se escogió este lenguaje por su portabilidad, eficiencia y disponibilidad de librerías[29][28]. Las librerías más importantes utilizadas son:

- Procesamiento de datos: *pandas* [36]
- Matemáticas y estadística: *numpy* [36]
- Gráficas y figuras: *matplotlib* [36]
- Aprendizaje de máquinas: *scikit-learn*[37].
- Grafos: *NetworkX* [38].

5.1 Agente

El agente (*logmapper-agent*) es un componente que se instala en cada nodo del sistema distribuido. Es el encargado de capturar las métricas del nodo y procesar en línea los archivos log, guardar la información en una estructura de datos que permite medir el rendimiento del componente, realizar el preprocesamiento en ventanas de tiempo y enviar la información al componente maestro.

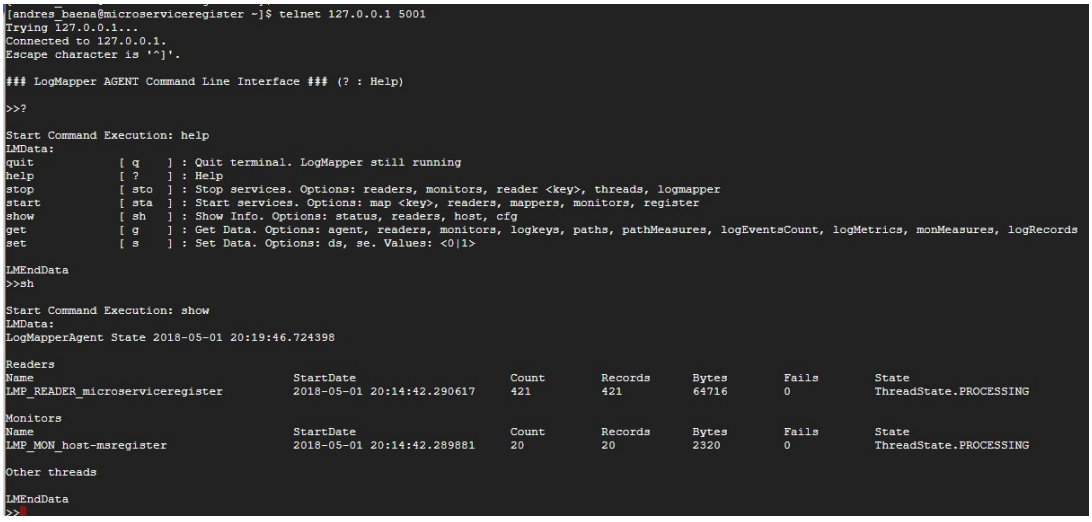


Figura 5-2: Interfaz de línea de comandos del logmapper-agent.

Se debe instalar un agente por cada nodo, para poder capturar la información de este. En cada agente se pueden configurar múltiples *readers* y múltiples *monitors*. Los *readers* son los componentes que procesan los archivos log y los guardan en una base de datos SQLITE. Los *monitors* son los componentes encargados de capturar datos del host o de componentes especializados tales como microservicios, servidores y bases de datos.

En tiempo de ejecución el agente se gestiona por medio de una interfaz de línea de comandos, a la cual se llega por medio de una conexión tipo telnet. En la **Figura 5-2** se muestra dicha interfaz, la ejecución del comando de ayuda y el informe de estado.

Los datos capturados por cada *reader* y cada *monitor* son almacenados en bases de datos SQLITE. Se manejan bases de datos independientes por cada fuente de datos. Esta modularización facilita el escalamiento de la herramienta y la independencia de cada proceso. Una característica importante de SQLITE es que puede operar en memoria, por lo que la velocidad de respuesta es muy alta. Sin embargo, manejar bases de datos en memoria genera inconvenientes a la hora de manejar diferentes hilos de ejecución. Para evitar este problema, en cada *reader* se utilizan dos bases de datos, una base de datos en memoria donde se procesa toda la información de los archivos log en línea, y otra base de datos persistente donde se guardan los datos agrupados por ventanas de tiempo. Esta última se maneja en archivo, ya que no requiere alta velocidad y de esta manera puede ser utilizada por otros hilos de ejecución. En la Figura 5-3 se muestra el modelo de datos manejado por el agente. La parametrización del agente, que en la figura se muestra en color gris, se maneja en un archivo de configuración. Los objetos de color amarillo y naranja corresponden a las tablas para el procesamiento de los archivos log, los cuales se manejan en la base de datos SQLITE en memoria. Finalmente, el objeto de color azul corresponde a las medidas del monitoreo del nodo.

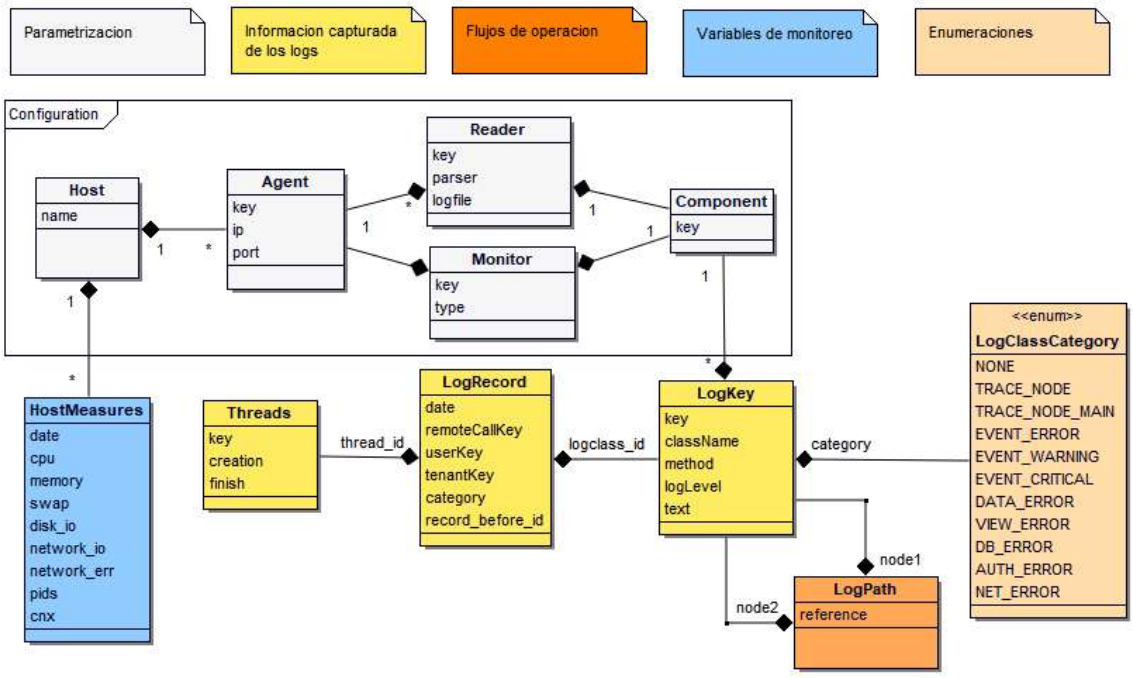


Figura 5-3: Modelo de datos del agente.

5.2 Maestro

El componente maestro (*logmapper-master*) es el componente que centraliza toda la información y realiza la medida del rendimiento y el análisis predictivo. Está compuesto por los siguientes módulos:

- Colector: este componente permite conectarse a cada agente y consultar información tal como la configuración de cada agente, los logKey y los logPath y si se quiere también las medidas de métricas y eventos.
- Server: servidor HTTP que expone métodos tipo REST para guardar los datos que son enviados desde los agentes. Este módulo permite recibir la información en línea de los agentes a medida que es generada.
- DAO (*Data Access Object*): es el módulo encargado de manejar el acceso a la base de datos PostgreSQL.

- *Performance*: es el componente encargado de medir el rendimiento.
- ML (*Machine Learning*): es el componente encargado de hacer el análisis predictivo. Prepara los datos, los transforma y realiza la predicción.

Al igual que el agente, este componente también tiene un archivo de configuración donde básicamente se configura la conexión a la base de datos. La configuración de la red de agentes y del sistema distribuido se almacena en la base de datos. El modelo de datos del *logmapper-master* se puede dividir en tres grupos. El primer grupo serían las tablas de configuración, las cuales se pueden ver en la **Figura 5-4** y contienen toda la configuración de la red de agentes y fuentes de datos.

- Lmp_host: host o nodos del sistema
- Lmp_agent: agentes habilitados en el sistema
- Lmp_component: componentes del sistema distribuido, son aquellos procesos ejecutables cuyo log es procesado y será objeto del análisis de rendimiento.
- Lmp_source: fuente de datos, corresponde a los *readers* y los *monitors* activos en los diferentes agentes.
- Lmp_measure_type: tipos de medidas. Permite gestionar dinámicamente la categoría y el tipo de transformación que se le da a los tipos de medidas.
- Lmp_measure_source: relaciona los tipos de medidas con las fuentes de datos. Permite configurar de manera independiente los tipos de medidas de cada fuente de datos.

El segundo grupo, que se puede ver en la **Figura 5-5**, es la información de los flujos de operación estructurada en *logKey* y *logPath*. Esta información fue creada por los agentes y centralizada en el maestro. Se tienen tres tablas:

- Lmp_logkey: LogKey de cada componente. La información del logKey es la que permite ubicar la falla de rendimiento en un punto especifico del código fuente.
- Lmp_logpath: Flujo de operación, refleja el paso de un logKey a otro. Las medidas de rendimiento se hacen sobre la duración de los logPaths.

- Lmp_remotecall: Son flujos de operación entre componentes. Son similares a los *logPath*, pero a diferencia de estos fueron inferidos por el maestro, y permiten crear relaciones entre componentes.

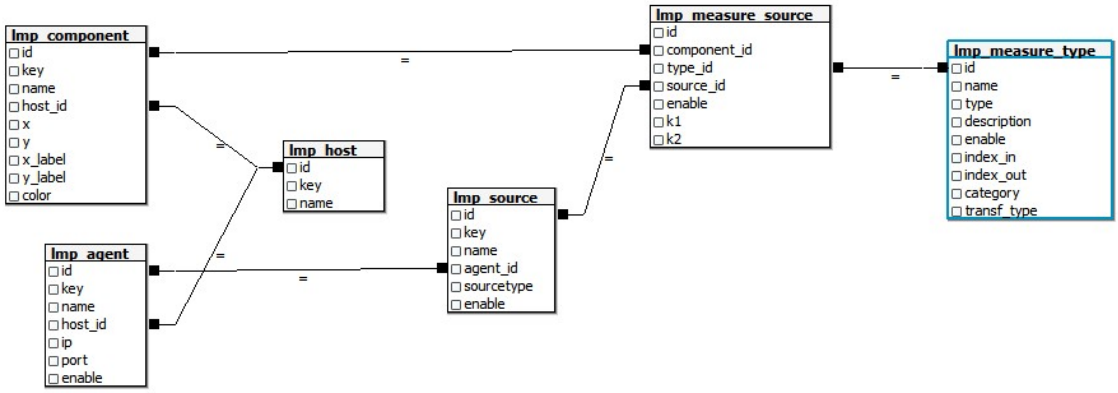


Figura 5-4: Tablas de configuración

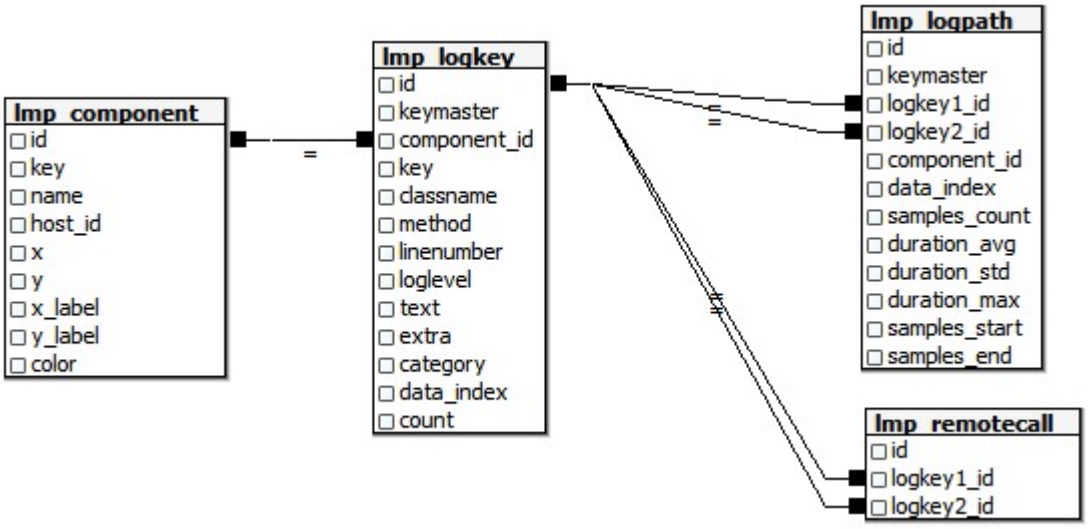


Figura 5-5: Tablas con flujos de operación.

El último grupo de tablas del maestro, mostrado en la **Figura 5-6**, corresponde a las tablas que almacenan las medidas capturadas y los resultados de la medida del rendimiento y el análisis predictivo. Las tablas son:

- Lmp_path_measure: medidas de los *logPath*. Son el insumo para la medida del rendimiento.
- Lmp_host_measure: medidas del nodo.
- Lmp_measure: medidas adicionales. Cada registro contiene solo un valor medido, el cual puede corresponder a eventos de los archivos log o datos creados desde los componentes de monitoreo especializados. Gracias a su estructura muy básica permite configurar nuevas medidas sin necesidad de cambiar el código fuente del sistema.
- Lmp_result: resultados de la medida de rendimiento y el análisis predictivo.

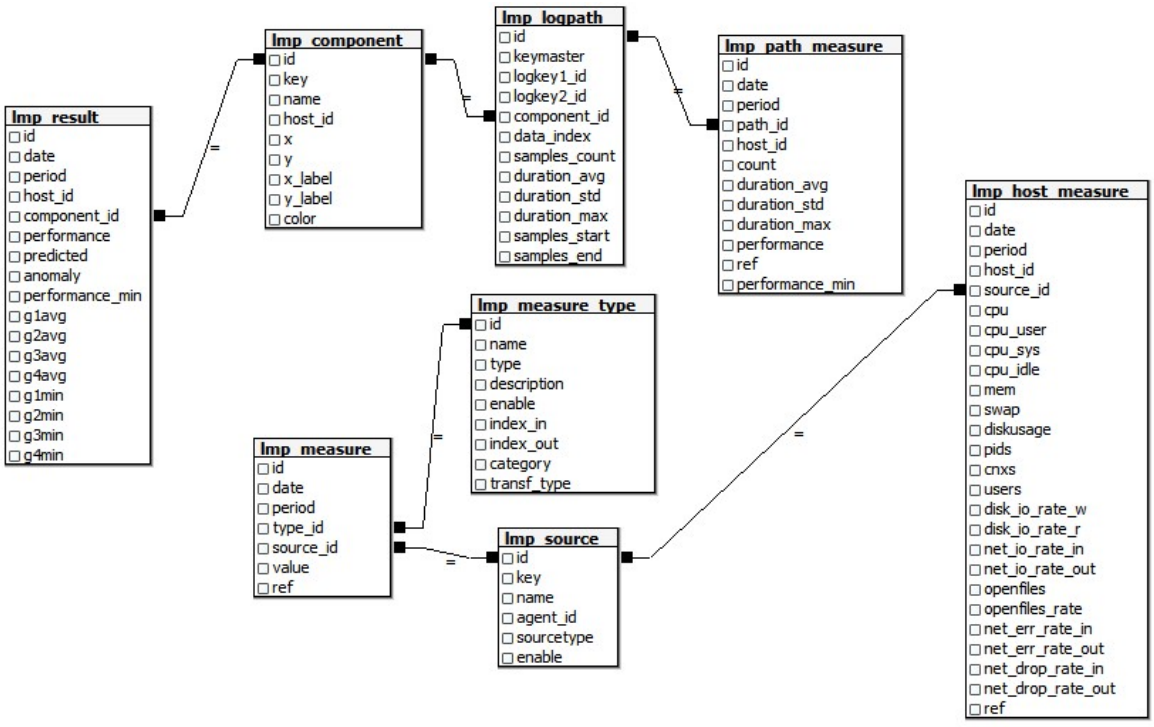


Figura 5-6: Tablas con datos de medidas.

5.3 Interfaz de usuario

El último componente de la herramienta es la interfaz de usuario (*logmapper-ui*), que es una aplicación de escritorio, que puede ser utilizada en plataformas Windows o Linux y que permite visualizar el estado de los diferentes componentes del sistema distribuido, incluyendo las medidas de rendimiento y las medidas capturadas desde los agentes. Este componente se conecta a la base de datos del componente maestro para visualizar la información. La interfaz tiene tres secciones principales:

- Un control tipo árbol, en la parte izquierda, donde aparecen los nodos, los componentes y las fuentes de datos. Para visualizar fácilmente los componentes con problemas de rendimiento se utilizan colores para enfocar la atención del usuario. En la columna rendimiento se muestran tres valores, los cuales corresponden al rendimiento promedio, rendimiento mínimo y el indicador promedio de anomalías.
- La sección superior derecha es el área de gráficas, donde se visualizan gráficamente los datos capturados. Las gráficas mostradas dependen del nodo seleccionado en el árbol. Si se selecciona el nodo raíz, se muestra un grafo dirigido cuyos nodos son los componentes del sistema distribuido y las aristas son los llamados entre ellos. Los nodos son mostrados con colores que reflejan el rendimiento así:
    - Negro: No hay medida de rendimiento en la ventana de tiempo
    - Verde: El rendimiento es óptimo
    - Amarillo: Advertencia, el rendimiento entre 0.7 y 0.5
    - Rojo: rendimiento mínimo menor a 0.5
- En la parte inferior se encuentran controles que permiten cambiar el estado de la herramienta y navegar por el historial de datos fácilmente.

En las figuras **Figura 5-7** a la **Figura 5-10** se muestran los datos presentados en la interfaz de usuario, según el tipo de nodo seleccionado en el árbol izquierdo.

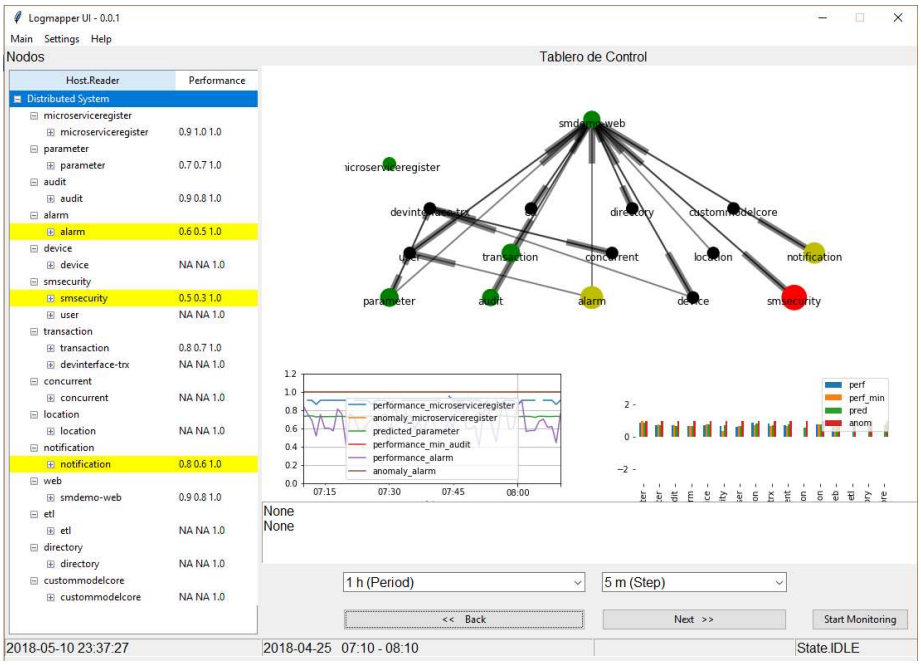


Figura 5-7: Interfaz de usuario al seleccionar el nodo raíz.

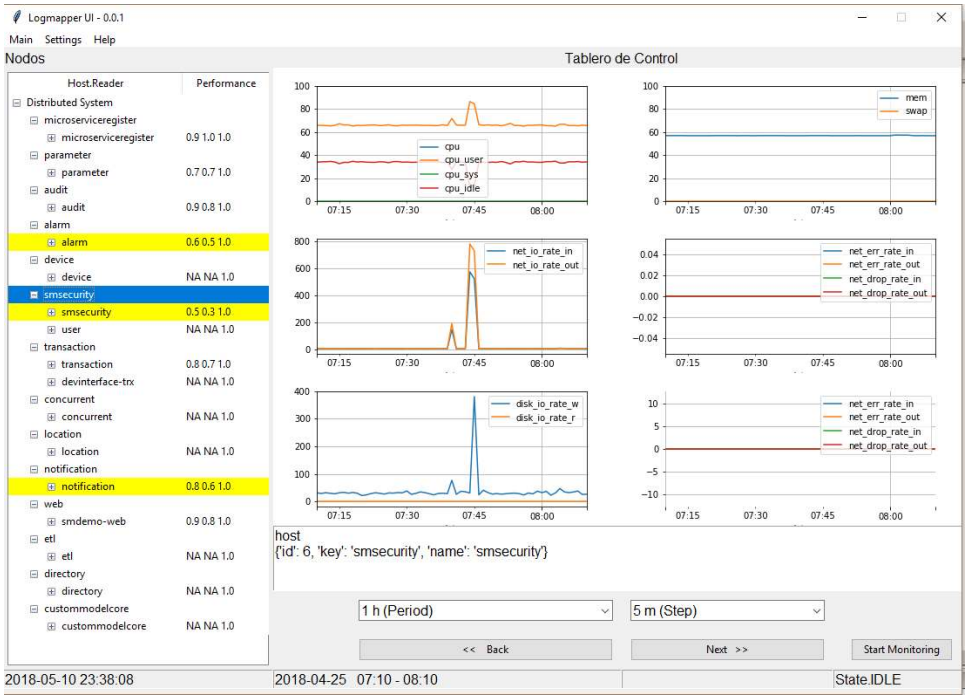


Figura 5-8: Interfaz al seleccionar un nodo.

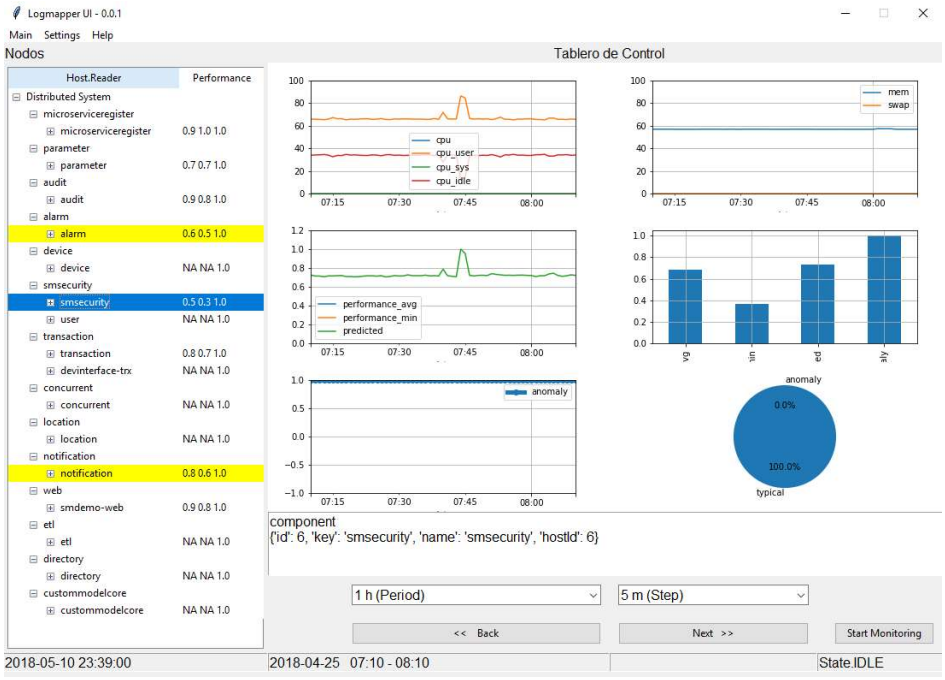


Figura 5-9: Interfaz al seleccionar un componente

Figura 5-10: Interfaz al seleccionar un monitor de microservicios.

6. Resultados

Para verificar la detección de problemas de rendimiento con la solución propuesta se despliega un ambiente de pruebas con un sistema distribuido donde se pueda emular la carga de un sistema en producción y generar diferentes fallas de rendimiento. El sistema distribuido utilizado es una aplicación llamada Siplex Management (SM), la cual es una plataforma de desarrollo para implementar rápidamente soluciones de gestión empresarial para diferentes tipos de negocio [39]. Esta plataforma se puede implementar de manera distribuida gracias a que está estructurada con una arquitectura de microservicios. Se escogió este sistema por la facilidad de acceso del investigador a este sistema y su código fuente, permitiendo recrear fallas y condiciones particulares. En la **Figura 6-1** se muestra la interfaz de usuario de esta aplicación.

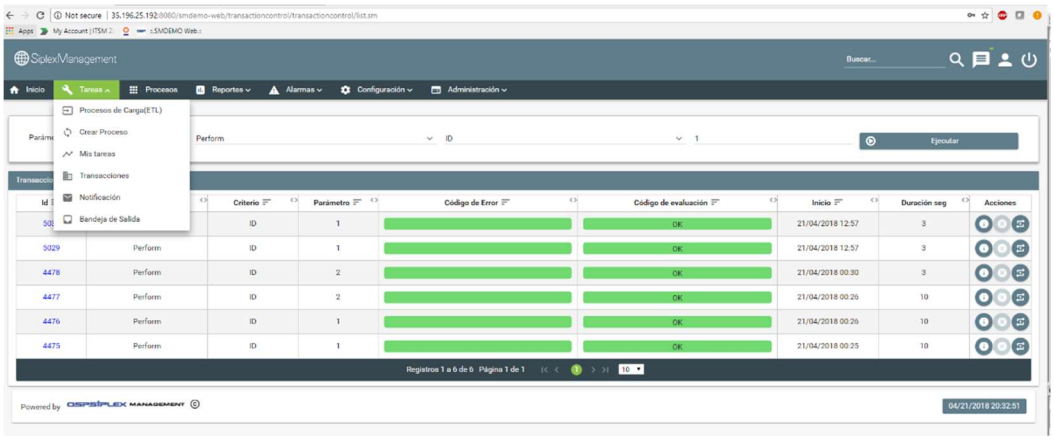


Figura 6-1: Interfaz de usuario aplicación SM.

El SM es un software construido en JAVA, utilizando como *framework* de desarrollo *SpringFramework*, el cual ofrece múltiples librerías que facilitan y organizan el desarrollo

de aplicaciones empresariales [40]. El sistema cuenta con diferentes microservicios, cada uno encargado de realizar una función específica, por ejemplo, el microservicio de seguridad se encarga de la autenticación en el sistema y los permisos de acceso a sus diferentes módulos. Los microservicios son un estilo de arquitectura de software que consiste en implementar aplicaciones utilizando un conjunto de pequeños servicios especializados, cada uno ejecutándose independientemente. Cada microservicio debe ser liviano, poco acoplado, enfocado en realizar una tarea específica, puede tener su propia base de datos y ofrece un mecanismo de comunicación abierto y liviano (por ejemplo servicios web tipo REST) [23].

En el SM, cada microservicio se conecta a una base de datos relacional tipo PostgreSQL [41] y ofrece una interfaz API a través de servicios web tipo REST. Entre las funciones disponibles por el microservicio se encuentra la posibilidad de consultar el estado del microservicio, ofreciendo métricas que son capturadas por la herramienta Logmapper para complementar el conjunto de datos a analizar. La interfaz WEB es una aplicación construida en Primefaces, el cual es una implementación de Java Server Faces [42], desplegada en un servidor Apache Tomcat. En la **Figura 6-2** se muestra una parte del diagrama de componentes de la aplicación SM y el Logmapper. La parte restante son los otros microservicios, los cuales tienen una configuración similar a los mostrados en la figura.

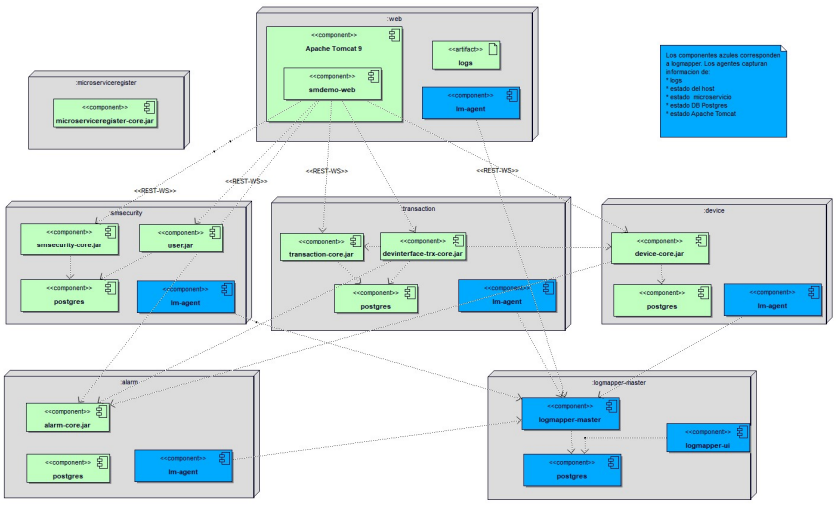


Figura 6-2: Diagrama de componentes parcial de la aplicación SM y LogMapper

6.1 Configuración ambiente de pruebas

La implementación del ambiente de pruebas está compuesta por 15 microservicios y una aplicación Web, desplegados en 15 nodos. Cada nodo incluye una base de datos PostgreSQL 10.2 de manera que los microservicios de un nodo interactúan con la base de datos que está en éste.

Para tener un ambiente distribuido se utilizó la plataforma *Google Cloud Platform* (GCP), donde se crearon las diferentes máquinas virtuales o instancias de *Google Compute Engine* (CGE) para cada uno de los nodos del sistema, en estos nodos se usó un sistema operativo Linux y diferentes especificaciones de hardware. GCP distribuye sus servicios en regiones y zonas, las regiones son ubicaciones geográficas donde se pueden ejecutar sus recursos. Cada región tiene una o más zonas. Por ejemplo, la región us-central1 denota una región en el centro de Estados unidos que tiene zonas us-central1-a, us-central1-b, us-central1-c, and us-central1-f [43]. En la **Figura 6-3** se muestra la consola de control de GCE, donde se gestionan las máquinas virtuales creadas.

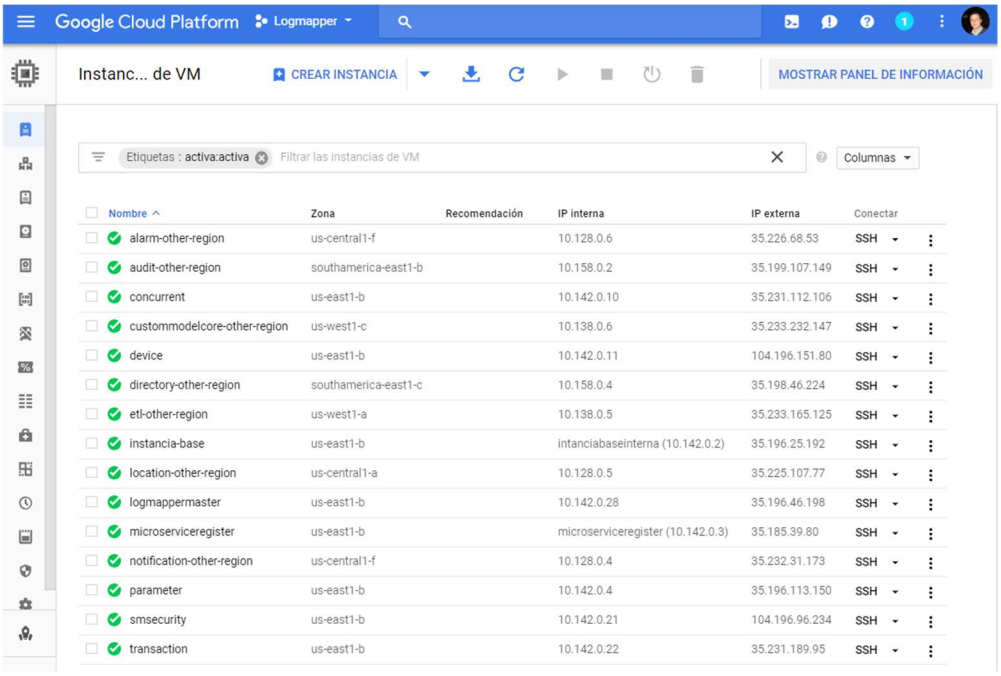


Figura 6-3: Consola de control de Google Compute Engine.

Se crearon 15 nodos, que se distribuyeron en diferentes regiones geográficas, 13 para microservicios, uno para la interfaz WEB y uno para el componente maestro del *LogMapper*. En la **Figura 6-4** se muestran los nodos desplegados y su distribución en regiones geográficas y en la **Tabla 6-1** se listan las características del hardware de cada nodo.

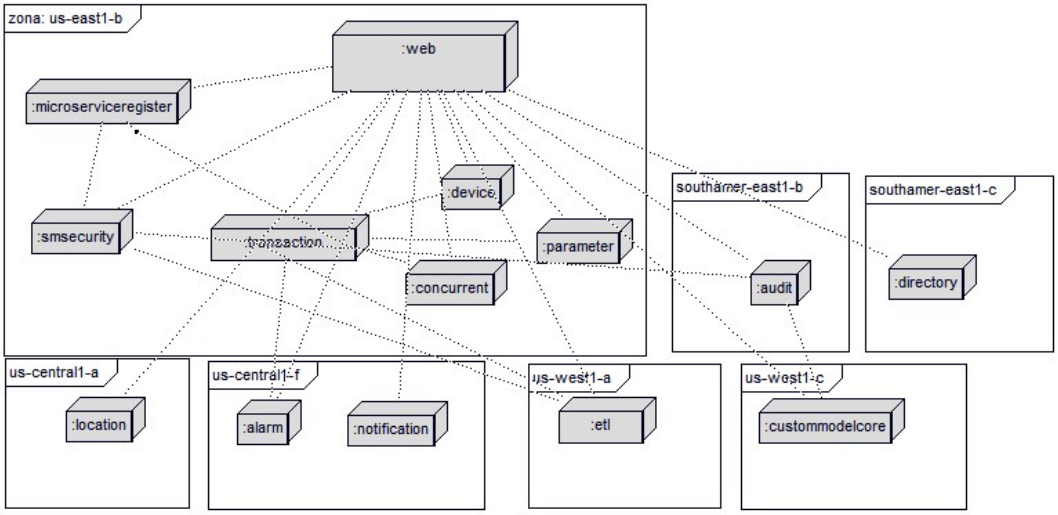


Figura 6-4: Nodos desplegados en GCP con sus regiones.

Tabla 6-1: Nodos ambiente de pruebas.

#	Nodo	Componentes	CPU	RAM
1	alarm-other-region	alarm-core	1 vCPU	1,25 GB
2	audit-other-region	audit-core	1 vCPU	1,25 GB
3	concurrent	concurrent-core	1 vCPU	2,5 GB
4	custommodelcore-other-region	custommodel-core	1 vCPU	1,5 GB
5	device	device-core	1 vCPU	1,5 GB
6	directory-other-region	directory-core	1 vCPU	1,25 GB
7	etl-other-region	etl-core	1 vCPU	1,25 GB
8	location-other-region	location-core	1 vCPU	1,25 GB
9	microserviceregister	microserviceregister-core	1 vCPU	3,75 GB
10	notification-other-region	notification-core	1 vCPU	1,25 GB
11	parameter	parameter-core	1 vCPU	2 GB
12	smsecurity	smsecurity-core user-core	1 vCPU	3 GB
13	transaction	transaction-core devinterface-core	1 vCPU	2,5 GB
14	WEB (instancia-base)	web (Tomcat)	2 vCPU	13 GB
15	logmappermaster	logmapper-master	1 vCPU	4 GB

6.2 Generación de carga

Para emular la operación del sistema por parte de múltiples usuarios se utiliza la aplicación Apache JMeter™, el cual es un aplicativo *opensource* diseñado en Java con el propósito de hacer pruebas de carga y medir el rendimiento de aplicaciones. Inicialmente fue diseñado para aplicaciones web, pero las versiones actuales permiten probar diferentes tipos de aplicaciones [44]. En **Figura 6-5** se muestra la interfaz de usuario de *Jmeter*.

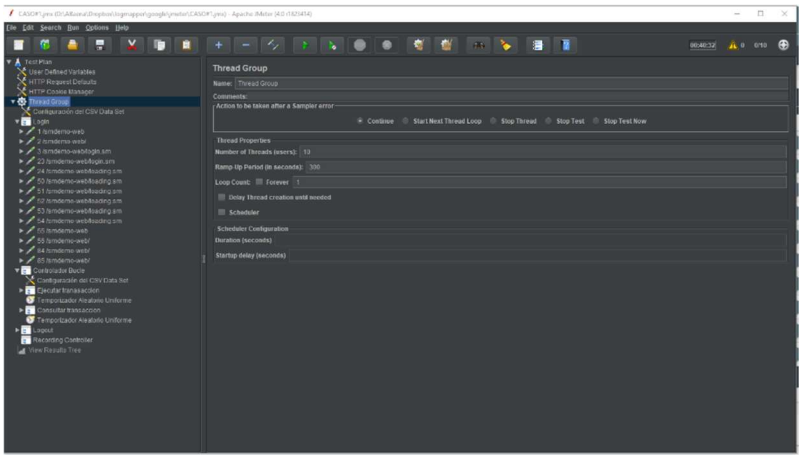


Figura 6-5: Interfaz Jmeter.

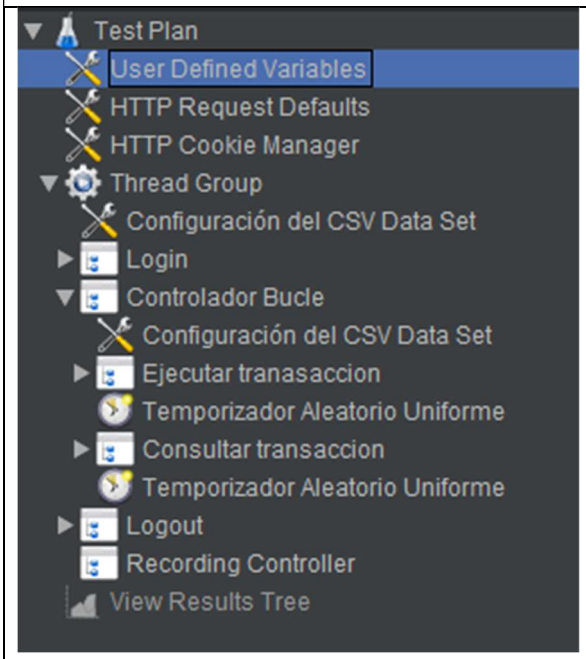
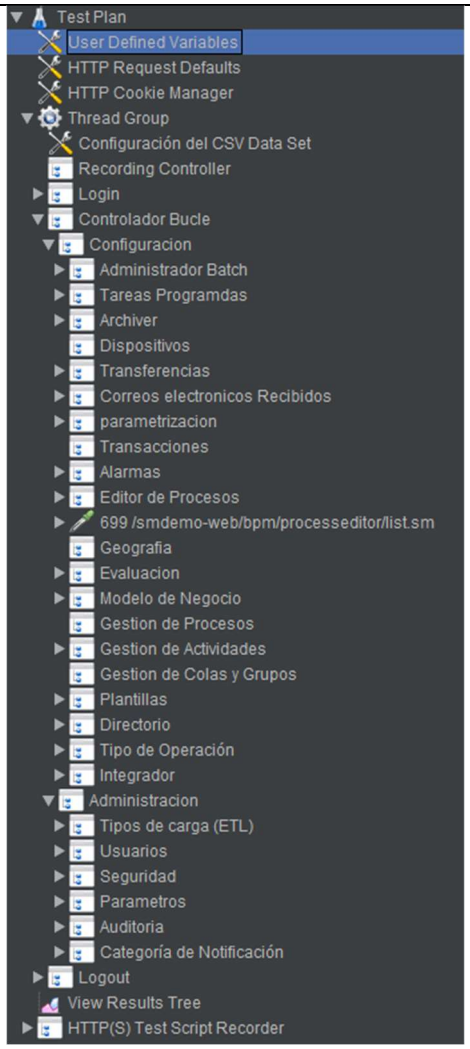
Se definieron dos planes de prueba con diferentes patrones de uso del sistema (Ver **Tabla 6-2**), los cuales permiten recrear de manera controlada una carga del sistema distribuido muy similar a un ambiente real. Cada plan de prueba puede emular múltiples usuarios concurrentes, que se autentican por medio de la interfaz web y navegan y desarrollan diferentes acciones sobre el sistema. Se usaron diferentes elementos de configuración que brinda la herramienta *Jmeter* para simular la carga que desempeñarían usuarios normales de la aplicación, algunos de los elementos más relevantes que se usaron fueron:

- CSV Data set: Para leer datos desde archivos CSV que permitieron variar los datos con los que se ejecutaban las pruebas (usuarios, passwords, parámetros).
- Temporizador Aleatorio Uniforme: Para agregar retardos de tiempos aleatorios entre la ejecución de las peticiones de los planes de pruebas, logrando así

reproducir los tiempos, aleatorios, que un usuario real demoraría para realizar las diferentes acciones simuladas.

- Extractor de expresiones regulares: Para simular los diferentes controles de seguridad por sesión que tiene la aplicación.
- Controlador de bucle: Para definir la cantidad de repeticiones que se harían del ciclo interno del plan de pruebas.

Tabla 6-2: Casos de prueba de *Jmeter* para emulación de carga

Caso de prueba #1	**Caso de prueba #2**
	

En todos los planes de prueba, se definieron parámetros que permitían delimitar la cantidad de ejecuciones que se harían sobre la aplicación:

- Número de hilos: Cantidad de usuarios simultáneos que ejecutan el plan de pruebas.
- Periodo de rampa: Tiempo durante el cual se reparte el inicio de la ejecución de los usuarios definidos en el número de hilos.
- Bucle (externo): Cantidad de repeticiones que se hacen de la ejecución plan de pruebas.
- Bucle (interno): Cantidad de repeticiones que se hacen de las peticiones enmarcadas dentro del Controlador de bucle, cantidad de veces que se quiere ejecutar el *loop* interno para un usuario del plan de pruebas.

Estos parámetros se definieron de manera que pudiesen establecerse al ejecutar el plan de pruebas desde el modo consola de *Jmeter*, ya que, para simular una gran cantidad de usuarios, la interfaz de usuario GUI demanda una gran cantidad de recursos de hardware, sobrecargando innecesariamente la máquina desde la cual se envían las peticiones. La **Figura 6-6** muestra la ejecución de un plan de pruebas desde el modo consola de *Jmeter*.

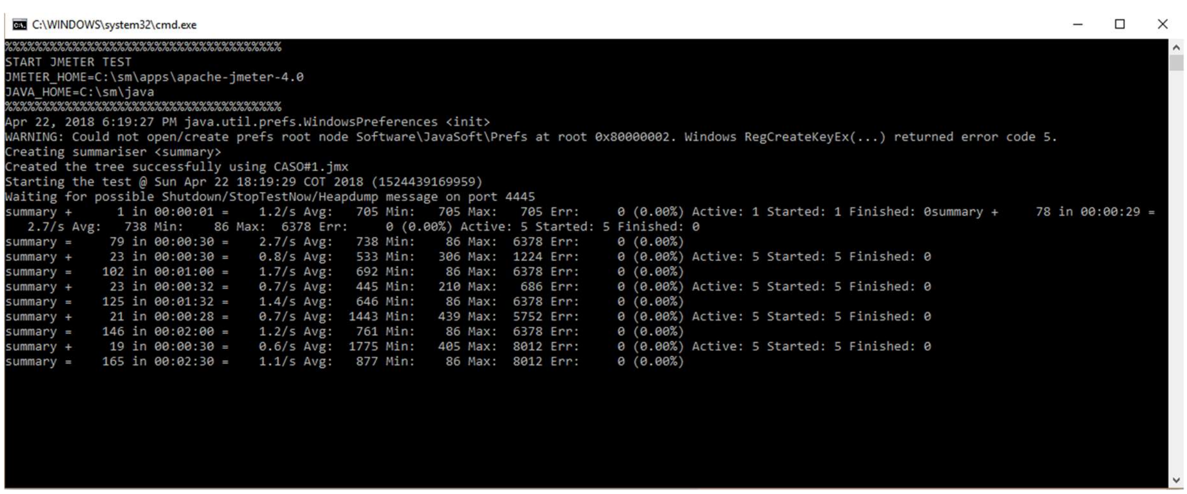


Figura 6-6: *Jmeter* ejecutado en modo consola.

Jmeter ofrece reportes HTML con diferentes gráficos que permiten examinar los resultados de la ejecución de un caso de prueba.

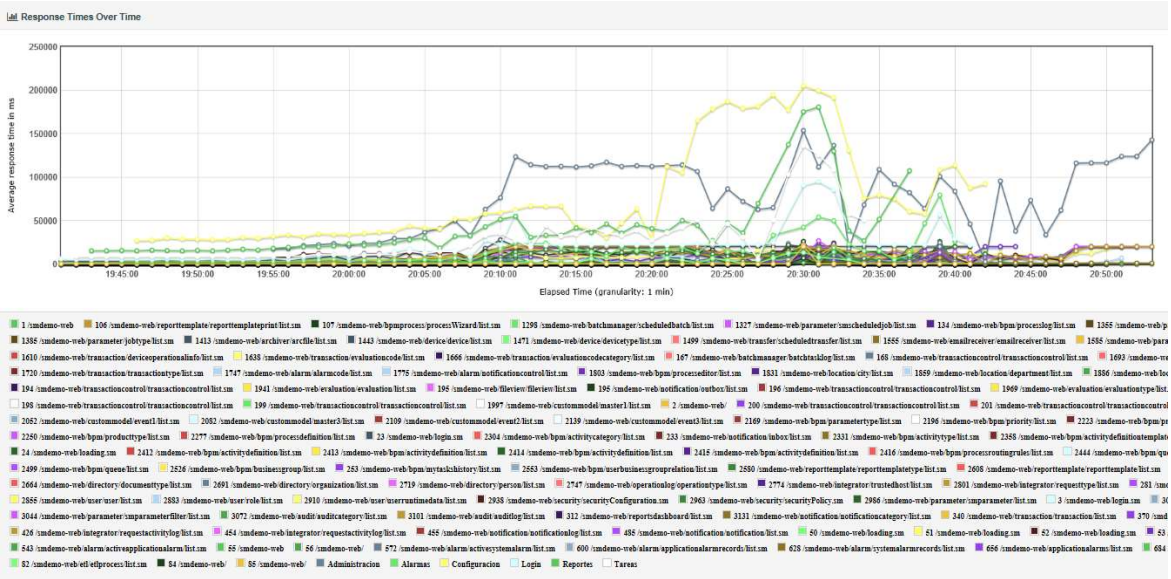


Figura 6-7: Reporte de *Jmeter* del tiempo de respuesta contra el tiempo.

La **Figura 6-6** muestra una de las gráficas de *Jmeter* para una corrida del caso de pruebas 2, el eje Y corresponde al tiempo promedio de respuesta a una petición HTTP y el eje X es el tiempo. En este reporte se tiene una línea por cada URL que es llamada por *Jmeter*, por lo tanto, si el rendimiento se comporta de una manera estable dichas líneas deben tender a permanecer horizontales. En la figura mostrada se observa que la mayoría de los tiempos de respuesta permanece estable, pero hay algunos casos en los que el tiempo de respuesta crece hasta cinco veces al valor inicial, lo que indica algún tipo de fallo. Herramientas como *Jmeter* permiten detectar que se presenta un problema en la respuesta de una URL, sin embargo, no ofrece información específica que pueda servir para un diagnóstico.

6.3 Detección de problemas de rendimiento

El ambiente de pruebas y la generación de carga expuestos anteriormente se utilizaron para emular un sistema distribuido en operación durante un periodo de cinco días. Se seleccionaron datos de aproximadamente 72 horas para realizar el entrenamiento y la validación de las técnicas de aprendizaje de máquinas, utilizadas para identificar el comportamiento típico del sistema. Los resultados de esta validación fueron expuestos en el apartado 4.4.5. En esta sección se exponen los resultados generales del monitoreo

realizado y los resultados obtenidos en la herramienta al inducir diferentes tipos de fallas que desembocan en problemas de rendimiento.

Las pruebas se realizaron durante un periodo de cinco días, donde se ejecutaron los planes de prueba explicados anteriormente con diferentes parámetros de número de hilos y duración. En la **Tabla 6-3** se muestran indicadores generales obtenidos de la base de datos del maestro. Una de las funciones principales de la herramienta es la capacidad de inferir los flujos de operación de un sistema distribuido, en la herramienta esto se puede evidenciar en el número de *logKeys* y *logPaths* creados por cada componente (**Tabla 6-4**) y visualmente se puede ver en el grafo de componentes del *logmapper-ui* (**Figura 6-8**).

Tabla 6-3: Indicadores generales de las pruebas realizadas.

Item	Cantidad
Host	14
Agentes	14
Componentes	16
Fuentes de datos	57
Tipos de medida	46
logKeys	1271
logPaths	3872
Llamados remotos	59
Medidas de path	882436
Medidas de host	66640
Otras medidas	932105
Lineas de log procesadas	27005103
MB procesados	5532

Tabla 6-4: *LogKeys* y *logPaths* creados por componente.

Componente	logKeys	logPaths
alarm	52	159
audit	30	72
concurrent	16	17
custommodelcore	20	69
device	41	121
devinterface-trx	48	71
directory	49	231
etl	47	124
location	54	252
microserviceregister	20	34
notification	49	187
parameter	65	356
smdemo-web	624	1495
smsecurity	41	95
transaction	64	409
user	51	180
TOTAL:	**1271**	**3872**

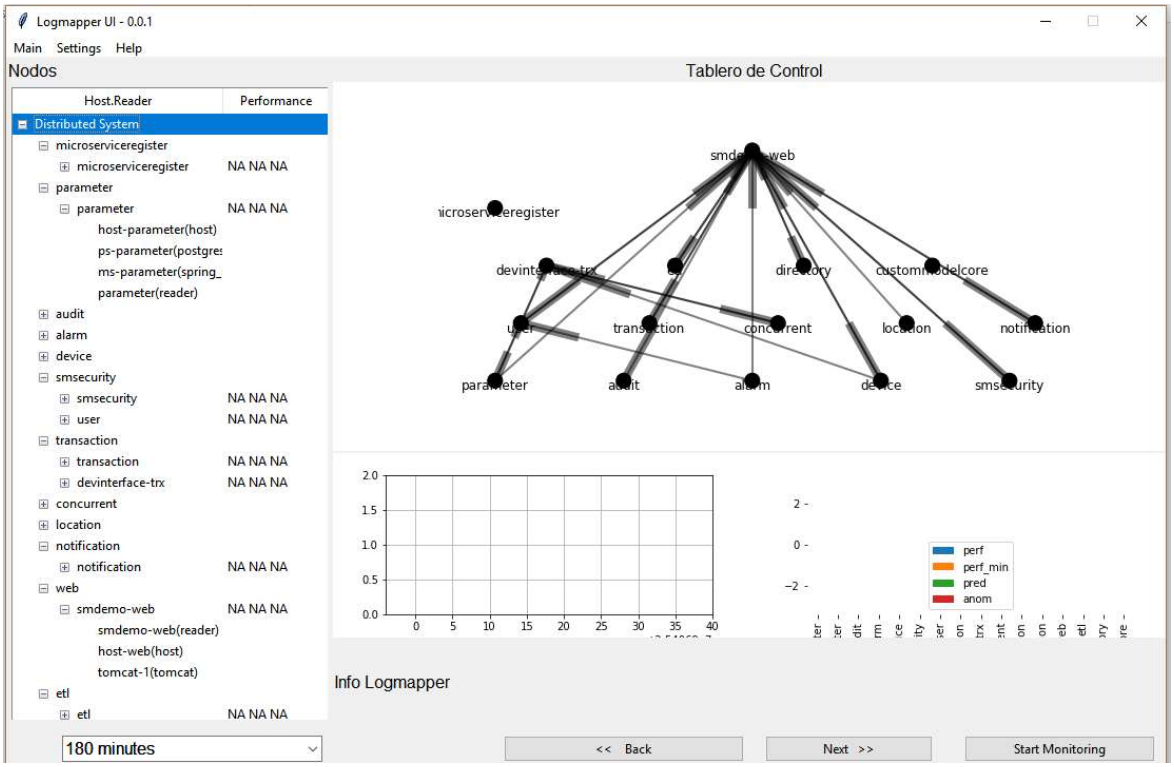


Figura 6-8: Árbol y grafo de componentes de LogMapper.

6.3.1 Medición del rendimiento del sistema distribuido en un estado normal

En esta prueba se verifican los reportes de la herramienta, manteniendo la aplicación en un estado típico. En la **Figura 6-9** se muestra el tiempo de respuesta vs el tiempo reportado por *Jmeter* y en la **Figura 6-10** el reporte general de LogMapper que indica que la aplicación está en un estado normal. Se muestran algunos nodos en amarillo, lo que indica que el rendimiento comienza a bajar, pero todavía se encuentra en un nivel aceptable.

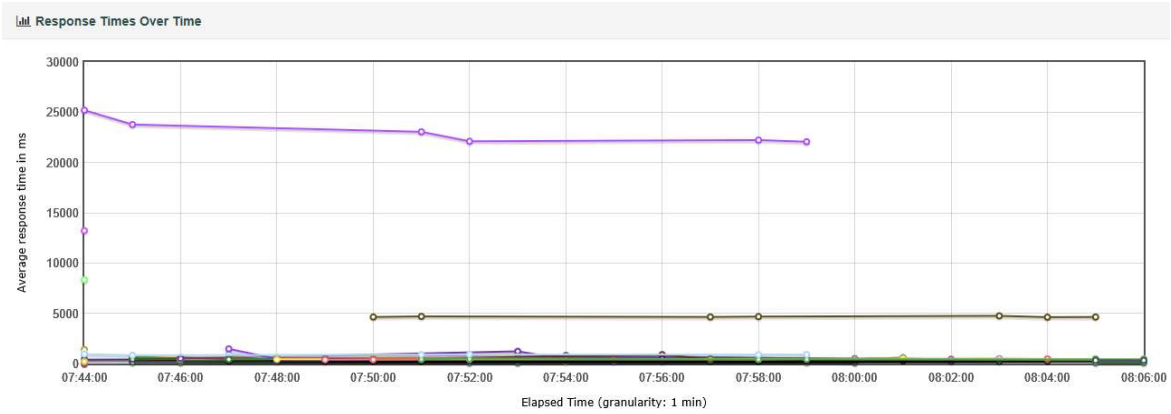


Figura 6-9: Reporte tiempo de respuesta Jmeter.

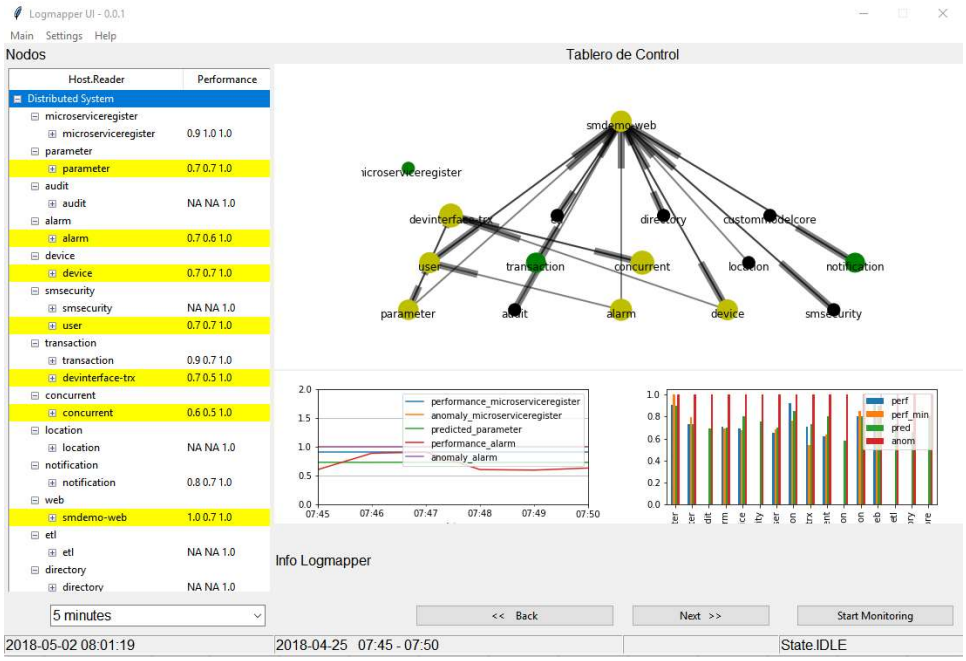


Figura 6-10: Reporte general LogMapper.

6.3.2 Detección de falla interna de un componente

En esta prueba se genera una falla de rendimiento en un componente interno del sistema distribuido. Para lograrlo, se modifica la aplicación a nivel de software, ingresando un retardo directamente en el código fuente, provocando que una transacción que antes demoraba 3 segundos pase a durar 30 segundos. Esta falla debe reflejarse en el

componente *devinterface-trx*. En la **Figura 6-11** se muestra el tiempo de respuesta vs el tiempo reportado por *Jmeter* para el caso de pruebas 1, en el cual no se observa afectación en los tiempos de respuesta. Esto es consistente con la falla que fue inducida, ya que el retardo se inyectó en la ejecución de una transacción y ésta se ejecuta asíncronamente y no es consultada directamente por el plan de pruebas.

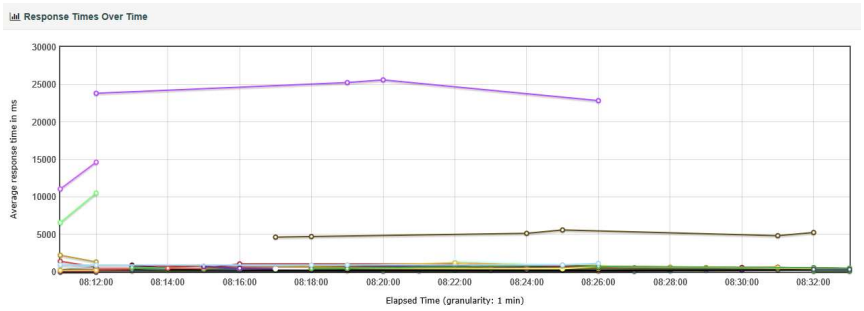


Figura 6-11: Reporte tiempo de respuesta Jmeter.

En el reporte general de LogMapper mostrado en la **Figura 6-12** se observa que tras la inducción del fallo, efectivamente se marca en rojo el componente *devinterface-trx*, lo cual es el primer paso para iniciar la tarea de diagnóstico. Del reporte del componente, mostrado en la **Figura 6-13** se pueden obtener varias conclusiones:

- La CPU y la memoria no presentan cambios importantes.
- En rendimiento mínimo llego a un valor de 0.3, muy lejos del rendimiento promedio y del esperado. Esto indica que algún proceso aislado está presentando fallas de rendimiento.
- No se indican anomalías en los factores no controlados del sistema.

Esto lleva a concluir que la falla no se trata por eventos externos al sistema, sino por una causa interna de la aplicación. Para analizar las causas internas la herramienta cuenta con un reporte que ofrece una lista con los *logPath* con menor rendimiento.

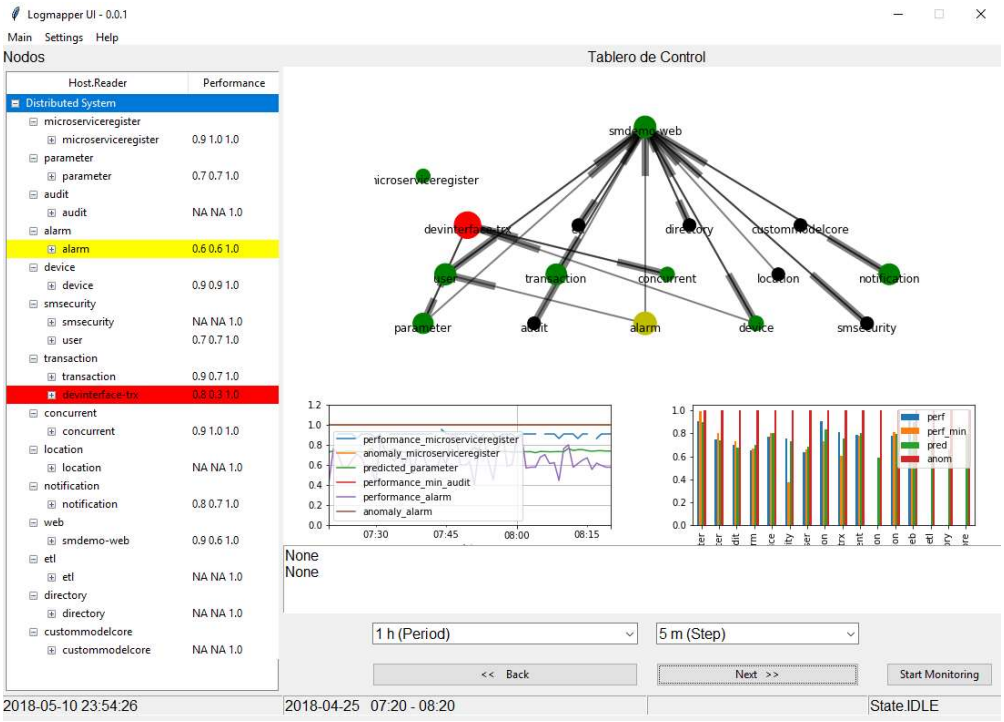


Figura 6-12: Reporte general LogMapper indicando componente en falla.

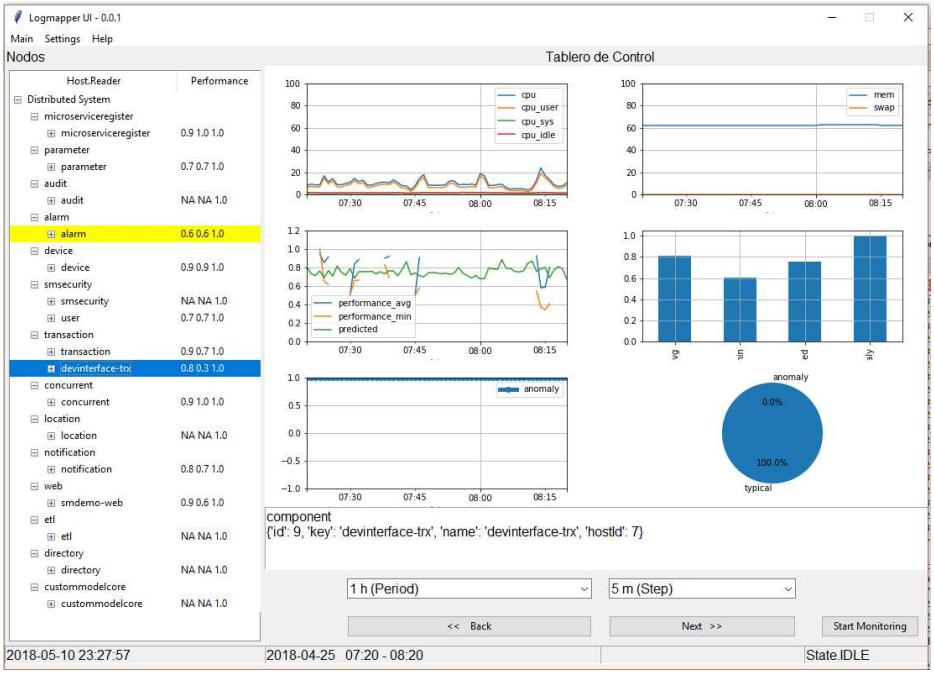


Figura 6-13: Reporte por componente LogMapper.

6.3.3 Detección de falla por factores externos

En esta prueba se ataca un componente de la aplicación llamado *concurrent*, lanzando directamente a este microservicio cientos de peticiones simultáneas. Este componente es utilizado a su vez por *devinterface-trx*, por lo que éste se debería ver afectado indirectamente. En la **Figura 6-14** se muestra el tiempo de respuesta vs el tiempo reportado por *Jmeter* para el caso de prueba 1, donde para la interfaz WEB no se nota ningún cambio significativo. Al igual que en el caso anterior, la capa de presentación no se afecta por el fallo de este componente.

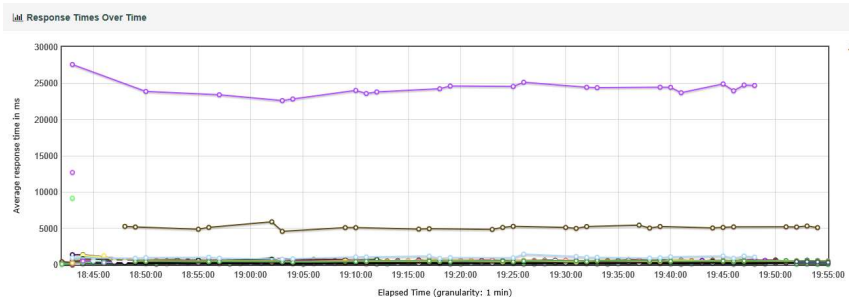


Figura 6-14: Reporte tiempo de respuesta Jmeter.

En el reporte general de LogMapper, mostrado en la **Figura 6-15** se muestra que la herramienta detecta que el componente *concurrent* presenta una baja de rendimiento y es marcado en rojo por la detección de un comportamiento atípico de las variables no controladas. En el reporte del componente, mostrado en la **Figura 6-16** se confirma visualmente la detección del estado atípico consistentemente a partir de las 18:48.

Para diagnosticar cuál es el comportamiento anómalo la primera opción es revisar el reporte del nodo. Al examinarlo se observa claramente (Ver **Figura 6-17**) que en el momento que se empieza a marcar el comportamiento atípico también se observa un aumento brusco en el uso de la CPU, el tráfico de la red y la escritura en disco de dicho nodo. De igual manera, se revisan las otras fuentes de datos, y en éstas se observa que el microservicio registró un incremento de las sesiones activas en este mismo periodo (Ver

Figura 6-18). Toda esta información ofrecida por la herramienta permite concluir que se trata de un comportamiento atípico del sistema, que no es generado por el software y que la causa raíz está relacionada con el incremento en las sesiones activas del microservicio.

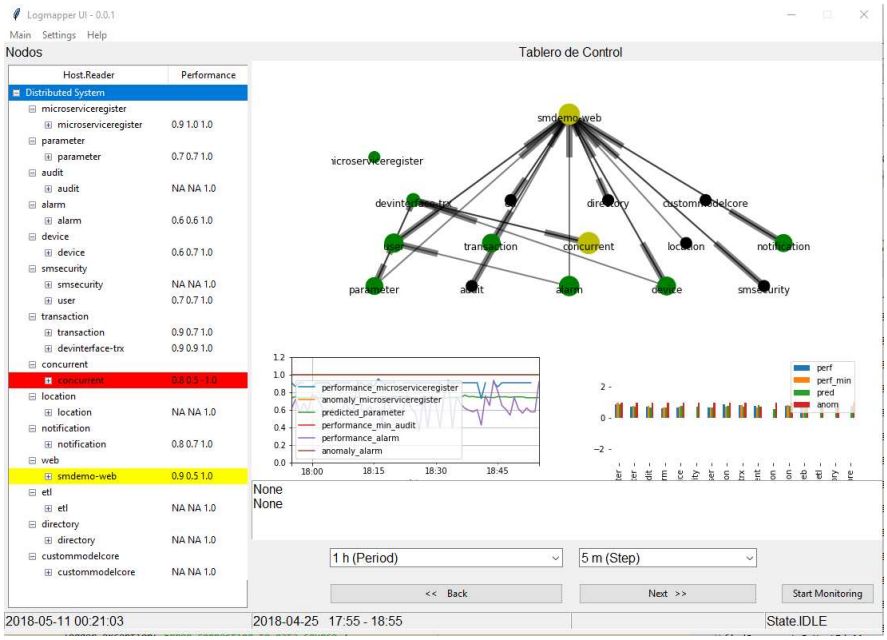


Figura 6-15: Reporte general de LogMapper.

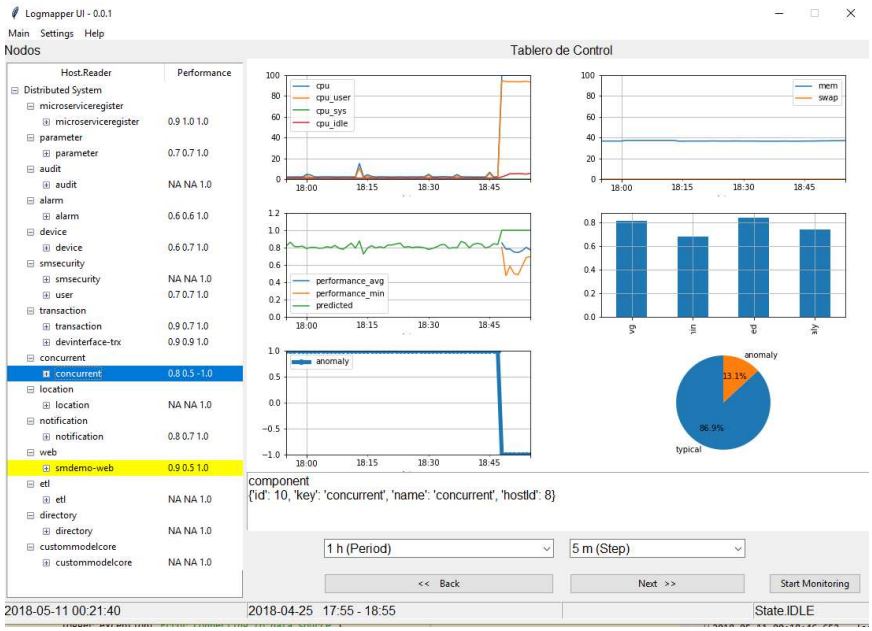


Figura 6-16: Reporte de componente *concurrent*.

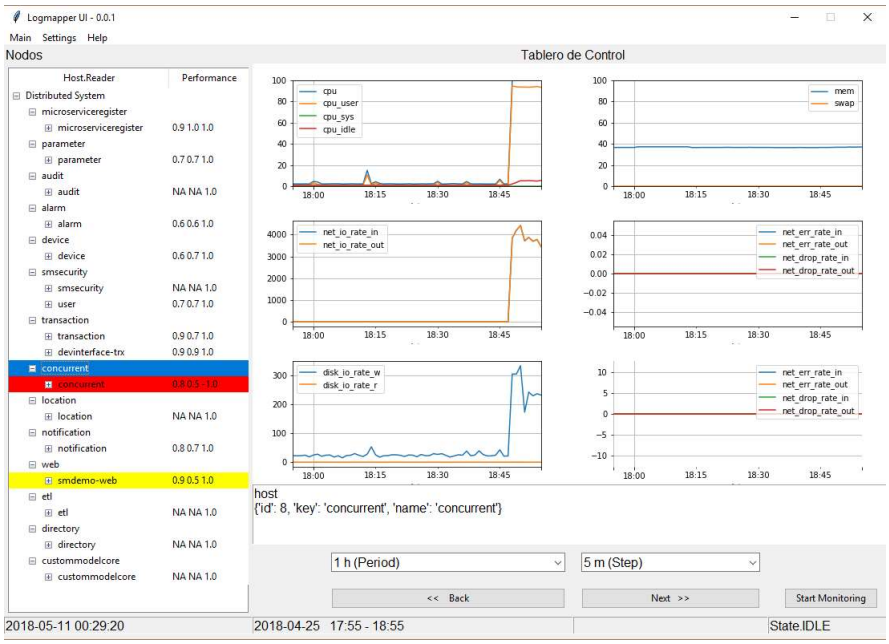


Figura 6-17: Reporte del nodo *concurrent.*

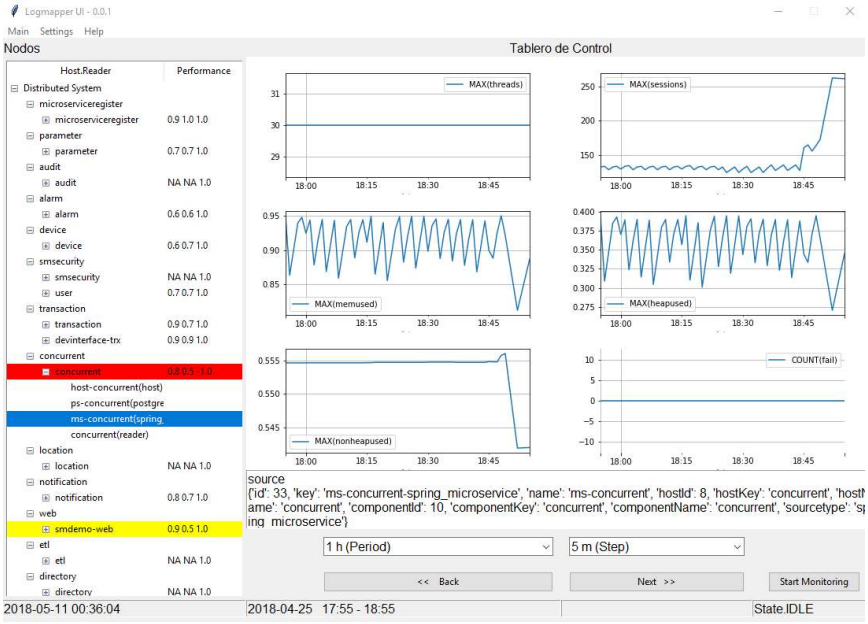


Figura 6-18: Reporte del microservicio *concurrent.*

6.3.4 Detección de falla por carga

En esta prueba se ejecuta el caso de prueba 1 de modo que se genere tráfico considerable sobre la interfaz web del sistema distribuido, el cual está centralizado en un solo componente denominado demo-web. En la **Figura 6-19** se muestra el tiempo promedio de respuesta, y se observa que el tiempo de respuesta para algunas peticiones presenta un incremento considerable desde el inicio de la prueba.

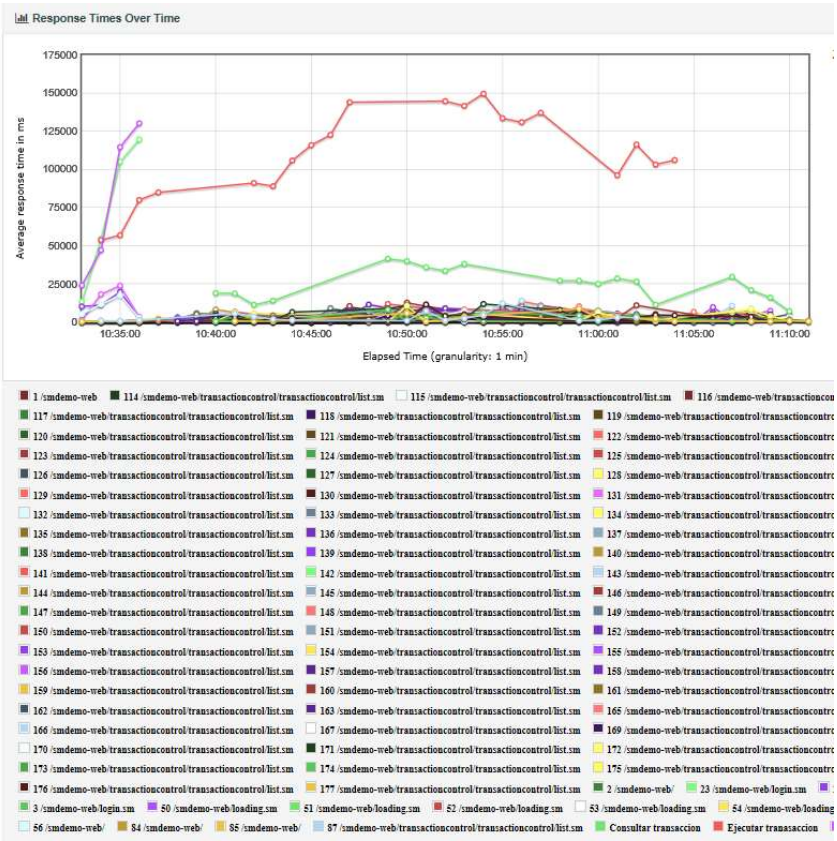


Figura 6-19: Reporte *Jmeter* con el tiempo de respuesta promedio.

Los resultados obtenidos por la herramienta en el reporte general (**Figura 6-20**) muestran una falla crítica del rendimiento en el componente smdemo-web, el cual llega a tener un valor mínimo de 0.0. También se observa que los componentes *smsecurity* y *alarm* presentan medidas de rendimiento mínimo que generan alerta y que otros 5 componentes presentan advertencia.

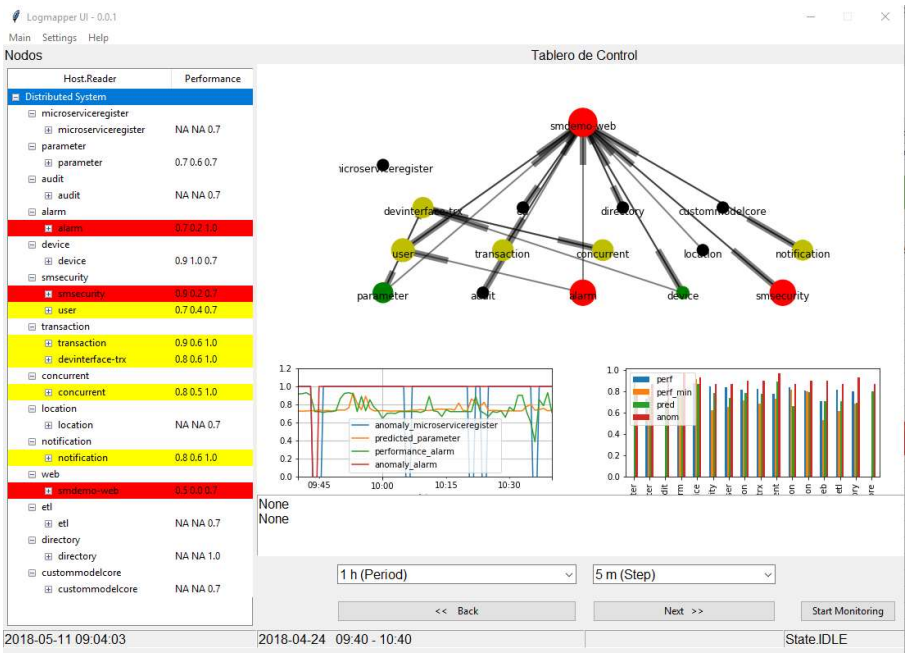


Figura 6-20: Reporte general LogMapper indicando estado del rendimiento de los componentes del sistema.

El reporte dado por la herramienta para el componente smdemo-web (**Figura 6-21**) muestra efectivamente un bajo en el rendimiento, un aumento del uso de la CPU, algunas anomalías detectadas, sin embargo, éstas no se presentan de forma continua. Al examinar las métricas de nodo (**Figura 6-22**) se observa el incremento en sus variables de CPU y tráfico de red, lo cual es consistente con el tráfico generado.

Para este caso, donde lo que se observa es una reducción del rendimiento por un aumento de carga, la información dada por la herramienta permite a los administradores del sistema distribuido identificar los nodos que presentan problemas de capacidad, para buscar estrategias tales como *tunning*, balanceo de cargas u otras que permitan aumentar la capacidad del sistema en general.

La herramienta ofrece reportes históricos, con diferentes escalas, que facilitan el análisis de toda la información capturada. En la **Figura 6-23** se puede ver el reporte histórico de toda la ejecución del caso de prueba, en el cual se observa gráficamente el periodo donde ocurrió el aumento de carga y la disminución del indicador de rendimiento y en la **Figura 6-24**, se observa cómo se comportaron las métricas del servidor Tomcat en este mismo periodo de tiempo.

Figura 6-21: Reporte del componente smdemo-web.

Figura 6-22: Reporte del nodo *web*.

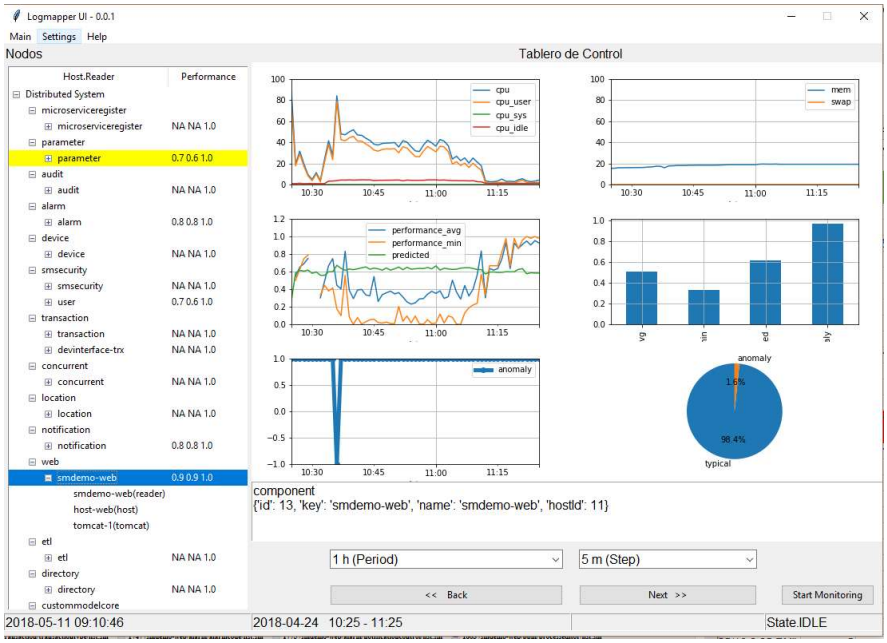


Figura 6-23: Reporte histórico del componente *smdemo-web.*

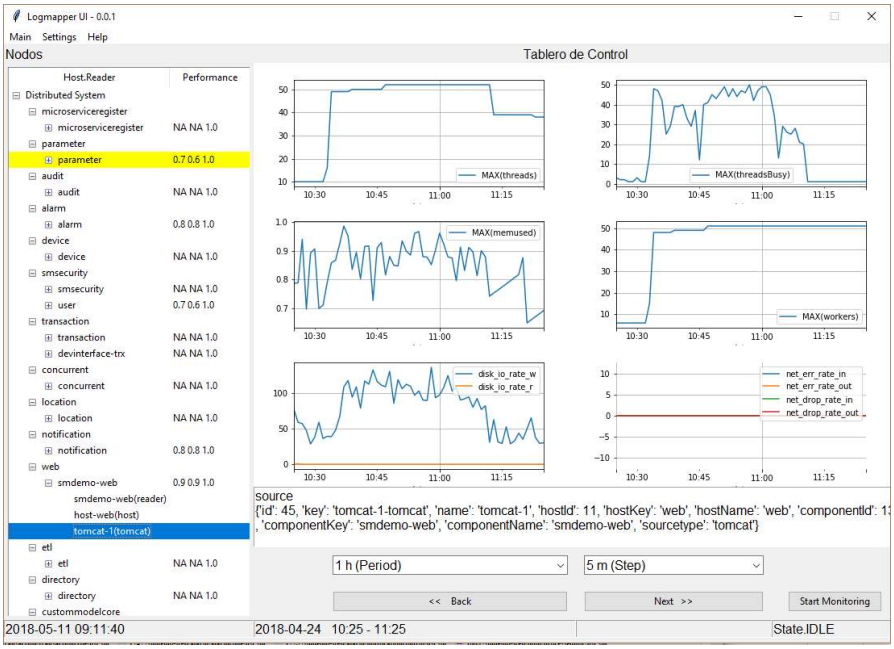


Figura 6-24: Reporte histórico del Tomcat que afecta a *smdemo-web.*

7. Conclusiones y recomendaciones

7.1 Conclusiones

Los resultados de la propuesta implementada permiten concluir que el modelo propuesto es una buena alternativa para analizar problemas de rendimiento en sistema distribuidos por las siguientes razones:

- Infiere automáticamente los llamados entre componentes y los flujos de operación y su duración promedio, lo cual permite analizar la dinámica interna de un sistema distribuido sin requerir un conocimiento profundo de todo el sistema.
- Captura información tanto de métricas como de eventos de diferentes fuentes, utilizando mecanismos tanto de caja negra como caja blanca, que ofrecen una visión más general de todos los factores que afectan el rendimiento de un sistema.
- El análisis predictivo, realizado con OCSVM y regresión, permite identificar rápidamente cuando un problema de rendimiento es causado por factores externos o internos al software
- La arquitectura distribuida y la utilización de Python mostró ser óptima para manejar gran cantidad de datos sin generar afectación al rendimiento del nodo y procesar datos en ventanas de tiempo de un minuto.
- Los reportes con alto contenido visual facilitan en gran medida la interpretación de la medida de rendimiento, y ayuda a focalizar de manera efectiva la causa raíz de un problema de rendimiento.
- La posibilidad de gestionar tipos de medidas independientes por cada componente le da a la herramienta una alta flexibilidad para realizar análisis sobre diferentes tipos de componentes y aplicaciones.

La estrategia propuesta ofrece una alternativa novedosa para diagnosticar el rendimiento con respecto a los trabajos previos, ya que involucra en una misma solución la inferencia de los flujos de operación y la duración de éstos, las métricas de los nodos y componentes y técnicas de aprendizaje de máquinas.

7.2 Recomendaciones

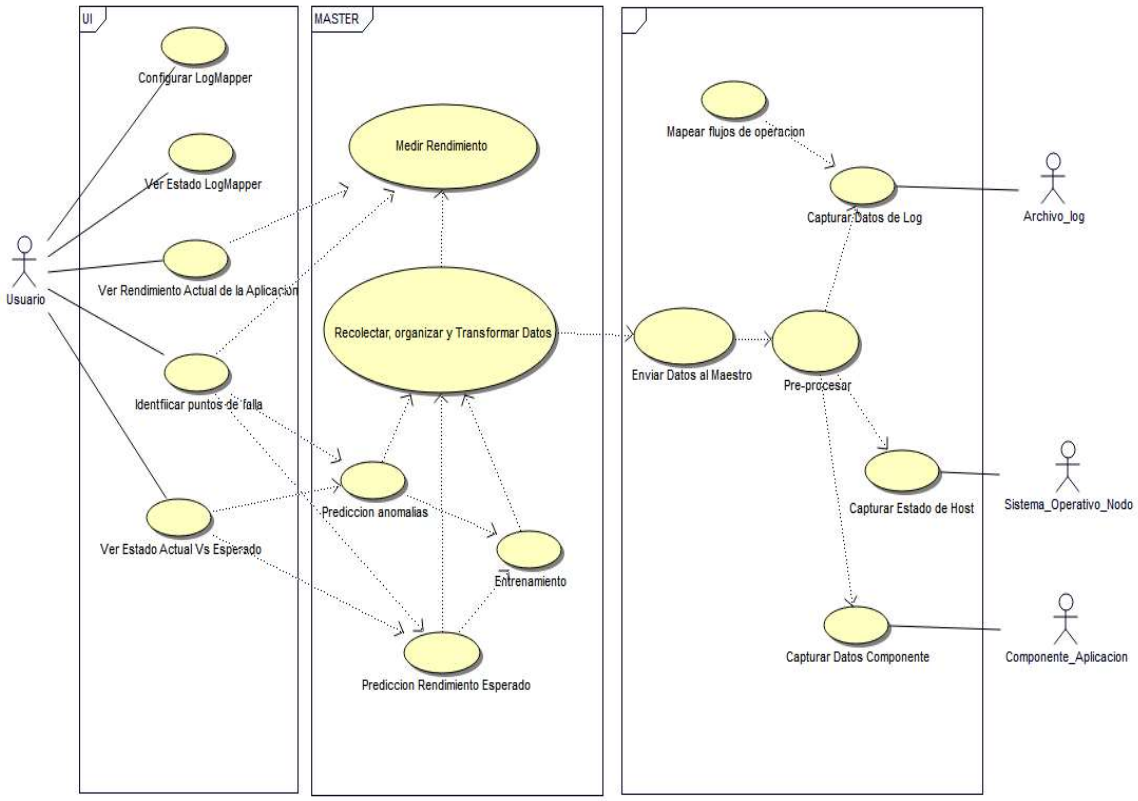
La herramienta LogMapper captura gran cantidad de datos que pueden aprovecharse de múltiples formas. Para el análisis de rendimiento se podrían agregar o quitar tipos de variables, también se podrían cambiar la forma de transformar los datos y los algoritmos de aprendizaje utilizados. También se podría utilizar la información capturada, o el mecanismo de captura y recolección para aplicar la herramienta a otro tipo de aplicaciones, como por ejemplo el análisis de seguridad.

Otra recomendación de trabajo futuro es aplicar la herramienta a otros sistemas distribuidos, de manera que se verifique su utilidad a diferentes aplicaciones.

Aunque se buscó desarrollar un software con buenas características de calidad, todavía es posible hacer muchas mejoras de configuración, rendimiento, usabilidad y seguridad, especialmente a los componentes maestro y la interfaz de usuario. El código fuente se deja disponible en Internet con la intención de que el proyecto continúe activo y en desarrollo.

A. Anexo: Casos de uso y requisitos no funcionales

Los casos de uso indican que va a realizar el software. A continuación, se resumen los casos de uso de la herramienta LogMapper, agrupados por el módulo que los implementa.



Casos de uso de la herramienta.

Actor	Descripción
Usuario	Usuario de la herramienta. Se encarga de configurar y operar la herramienta para detectar problemas de rendimiento
Archivo log	Archivos de log de la aplicación.
Sistema operativo nodo	Sistema operativo de cada host del sistema distribuido. Este actor ofrece las métricas del host.
Componente aplicación	Componente de la aplicación que puede ser monitoreado para obtener métricas y eventos.

Actores de la herramienta referenciados en los CU.

ID	Nombre	Descripción
CU001	Configurar LogMapper	El sistema se debe poder configurar a través de archivos de configuración o registros en base de datos
CU002	Ver estado LogMapper	La interfaz de usuario debe permitir visualizar el estado de la captura de datos
CU003	Ver rendimiento actual del sistema distribuido	Ver gráficamente el rendimiento del sistema distribuido. Se deben tener vistas globales del sistema y por componente
CU004	Identificar puntos de falla	Los reportes gráficos deben permitir resaltar los posibles puntos de falla.

Casos de uso de la interfaz de usuario.

ID	Nombre	Descripción
CU101	Recolectar, organizar y transformar datos	Recolectar los datos de los diferentes agentes, organizarlos y transformarlos en la estructura requerida para poder medir el rendimiento y realizar las predicciones
CU102	Medir rendimiento	Medir el rendimiento de cada componente basado en los datos obtenidos. Este caso de uso incluye el cálculo de los tiempos de referencia
CU103	Entrenamiento	Este caso de uso se encarga de crear la matriz de datos de entrenamiento, realizar el proceso de aprendizaje y hacer la validación del modelo
CU104	Predicción anomalías	Se realiza la predicción de anomalías con los datos recolectados y se guarda el registro histórico
CU105	Predicción rendimiento esperado	Se realiza la predicción del rendimiento esperado con los datos recolectados y se guarda el registro histórico

Casos de uso el módulo maestro.

ID	Nombre	Descripción
CU201	Capturar datos de log	Procesar en línea los archivos log y almacenarlos en una estructura de datos.
CU202	Mapear flujos de operación	Inferir los flujos de operación de modo que se creen los logPaths necesarios para poder medir el rendimiento.
CU203	Capturar estado del host	Capturar métricas y eventos del nodo utilizando la información entregada por el sistema operativo.
CU204	Capturar datos componente	La aplicación puede tener varios componentes ejecutables que pueden ser consultados para obtener datos que pueden influir en el rendimiento.
CU205	Preprocesar datos	Agrupar los datos en ventanas de tiempo.
CU206	Enviar datos al maestro	A medida que se tengan datos, estos deben ser enviados al maestro.

Casos de uso del módulo agente.

Requisitos no funcionales

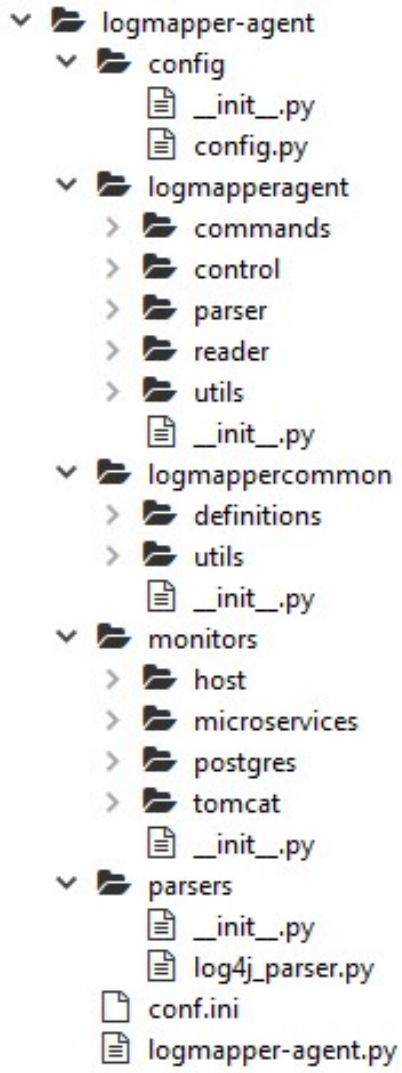
Los requisitos no funcionales especifican restricciones y características de calidad que deben tenerse en cuenta al momento de diseñar la arquitectura de la herramienta.

ID	Nombre	Descripción
NFR001	La herramienta debe generar una carga mínima sobre el sistema distribuido monitoreado.	
NFR002	La herramienta debe ser escalable junto con el sistema distribuido.	
NFR003	La herramienta debe poder ejecutarse en múltiples plataformas.	
NFR004	La herramienta debe permitir agregar nuevos parser de una manera sencilla, modificando solo la configuración.	
NFR005	La herramienta debe entregar información clara para el usuario final.	

Requisitos no funcionales.

B. Anexo: Paquetes y configuración logmapper-agent

El código fuente está estructurado en paquetes, los cuales se describen a continuación.



Paquete	Descripción
config	Manejo del archivo de configuración y utilidades relativas a éste.
logmappercommon.definitions	Módulo con constantes y parámetros fijos del código. Definición enumeración de categoría de eventos
logmappercommon.utils	Utilidades de uso general como logging, conectores a base de datos.
logmapperagent.commands	Implementación de comandos para la interfaz de línea de comandos.
logmapperagent.control	Módulos para la interfaz de comandos.
logmapperagent.parser	Módulos base para implementar los *parser*.
logmapperagent.reader	Módulos necesarios para el procesamiento de los archivos log y el mapeo de los flujos de operación.
logmapperagent.util	Utilidades para el agente.
monitor.host	Módulo de monitoreo para el estado del nodo.
monitor.microservices	Módulo de monitoreo para el estado de los microservicios.
monitor.postgres	Módulo de monitoreo para la base de datos PostgreSQL.
monitor.tomcat	Módulo de monitoreo para Apache Tomcat.

La configuración se hace por medio de un archivo de configuración, donde se especifican los parámetros generales y la lista de *monitors* y *readers* que se van a habilitar.

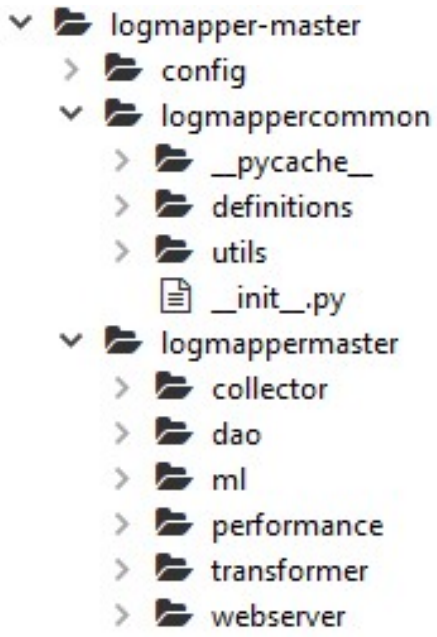
```
[LOGMAPPER]
dirbase = /logmapper/
dirlog = /logmapper/log/
dirdata = /logmapper/data/
logger.file = logmapper-agent.log
logger.debug.level = INFO
agent.key = agent-1
agent.ip = 127.0.0.1
agent.port = 5001
master.ip = 127.0.0.1
master.port = 5005
master.senddata.enable = False
agent.test.mode = True

[GENERAL]
hostname = localhost

[READERS_LIST]
readers.enable = True
readers.list = microserviceregister,device

[COMPONENT_MONITORS_LIST]
monitors.enable = True
monitors.list = host-app,ms-device,ps-app
```

Cada *reader* tiene una configuración independiente, en la cual se especifica el tipo de *parser* y la ruta del archivo que se va a leer.

```
[READER_device]
reader.enable = True
reader.key = device
reader.component = device
reader.sourcefilepath = /sm/log/device-core.debug.log
reader.parser.modulename = log4j_parser
reader.parser.classname = Log4jParser
```

C. Anexo: Paquetes logmapper-master

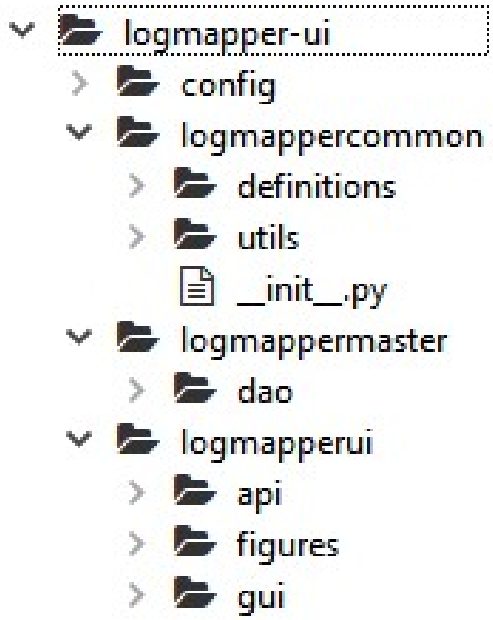
El código fuente está estructurado en paquetes, los cuales se describen a continuación:



Paquete	Descripción
config	Manejo del archivo de configuración y utilidades relativas a este.
logmappercommon.definitions	Módulo con constantes y parámetros fijos del código. Definición enumeración de categoría de eventos.
logmappercommon.utils	Utilidades de uso general como logging, conectores a base de datos.
logmappermaster.collector	Centraliza toda la información, realiza consultas a todos los agentes para crear la base de datos del maestro.
logmappermaster.dao	Módulo de acceso a datos. Ofrece los métodos de consulta, creación y actualización de las tablas de la herramienta.
logmappermaster.ml	Módulo de análisis predictivo. También incluye los módulos necesarios para hacer el entrenamiento y la validación.
logmappermaster.performance	Mide el rendimiento.
logmappermaster.transformer	Utilidades de transformación de datos.
logmappermaster.webserver	Servidor WEB para recibir en línea datos desde los agentes.

D. Anexo: Paquetes logmapper-ui

El código fuente está estructurado en paquetes, los cuales se describe a continuación.



Paquete	Descripción
config	Manejo del archivo de configuración y utilidades relativas a este.
logmappercommon.definitions	Módulo con constantes y parámetros fijos del código. Definición enumeración de categoría de eventos
logmappercommon.utils	Utilidades de uso general como logging, conectores a base de datos.
logmappermaster.dao	Módulo de acceso a datos. Ofrece los métodos de consulta, creación y actualización de las tablas de la herramienta.
logmapperui.api	Capa de abstracción que obtiene los datos requeridos por la interfaz de usuario
logmapperui.figures	Módulos que crean las diferentes gráficas utilizadas en la herramienta.
logmapperui.gui	Interfaz de usuario.

E. Anexo: Diccionario de Datos

Tabla:	**lmp_agent**	Agentes de la herramienta LogMapper	
Nombre	**Tipo**	**Nulos**	**Descripción**
id	integer	Si	Llave primaria
key	text	No	Llave o código único
name	text	No	Nombre descriptivo
host_id	integer	No	Id de la tabla host
ip	text	No	Direccion IP
port	integer	No	Puerto TCP
enable	boolean	No	Habilitado
Tabla:	**lmp_component**	Componente del sistema distribuido.	
Nombre	**Tipo**	**Nulos**	**Descripción**
id	integer	Si	Llave primaria
key	text	No	Llave o código único
name	text	No	Nombre descriptivo
host_id	integer	No	Id de la tabla host
x	real	No	Posición X en el grafo
y	real	No	Posición Y en el grafo
x_label	real	No	Posición X de la etiqueta en el grafo. (Reservado)
y_label	real	No	Posición X de la etiqueta en el grafo. (Reservado)
color	text	No	Color en el grafo (Reservado)
Tabla:	**lmp_host**	Host o nodo del sistema distribuido	
Nombre	**Tipo**	**Nulos**	**Descripción**
id	integer	Si	Llave primaria
key	text	No	Llave o código único
name	text	No	Nombre descriptivo
Tabla:	**lmp_source**	Fuente de datos	

Nombre	Tipo	Nulos	Descripción
id	integer	Si	Llave primaria
key	text	No	Llave o código único
name	text	No	Nombre descriptivo
agent_id	integer	No	Id del agente
sourcetype	text	No	Tipo de fuente de datos: SOURCE_TYPE_READER = "reader" SOURCE_TYPE_HOST = "host" SOURCE_TYPE_SPRINGMICROSERVICE = "spring_microservice" SOURCE_TYPE_TOMCAT = "tomcat" SOURCE_TYPE_POSTGRES = "postgres"
enable	boolean	No	Habilitado
Tabla:	**lmp_measure_type**	Tipo de medida	
Nombre	**Tipo**	**Nulos**	**Descripción**
id	integer	Si	Llave primaria
name	text	No	Nombre descriptivo
type	text	No	Tipo de medida. DATATYPE_PATH_METRICS = 'pathmet' DATATYPE_LOG_EVENTS = 'logeve' DATATYPE_LOG_METRICS = 'logmet' DATATYPE_MONITOR_HOST = 'host' DATATYPE_MONITOR_MICROSERVICE = 'ms' DATATYPE_MONITOR_TOMCAT = 'tomc' DATATYPE_MONITOR_POSTGRES = 'psql'
description	text	No	Descripción
enable	boolean	No	Habilitado
index_in	integer	No	Indice en datos de entrada
index_out	integer	No	Indice para tabla de salida
category	integer	No	categoria de la medida MEASURE_CAT_METRIC=0 MEASURE_CAT_EVENT=1
transf_type	integer	No	Tipo de transformación aplicada TRANSF_TYPE_NONE=0 TRANSF_TYPE_MINMAX=1 TRANSF_TYPE_STD=2 TRANSF_TYPE_PERCENTAGE=3
Tabla:	**lmp_measure_source**	Relacion entre tipo de medida y fuente de datos	

Nombre	Tipo	Nulos	Descripción
id	integer	Si	Llave primaria
component_id	integer	Si	Id del componente
type_id	integer	Si	Id del tipo de medida
source_id	integer	Si	Id de la fuente de datos
enable	boolean	No	habilitado
k1	real	No	Constante 1 para transformación
k2	real	No	Constante 2 para transformación
Tabla:	**lmp_logkey**	logKey: corresponde a un mensaje del archivo log	
Nombre	**Tipo**	**Nulos**	**Descripción**
id	integer	Si	Llave primaria
keymaster	text	Si	Llave o código único en el maestro
component_id	integer	No	Id del componente
key	text	No	Llave o código único en el agente
classname	text	No	Clase
method	text	No	Metodo
linenumber	text	No	Numero de linea
loglevel	text	No	Nivel del registro de log: DEBUG, INFO, WARN, ERROR
text	text	No	Texto del mensaje log
extra	text	No	Datos adicionales
category	text	No	NONE = 0 TRACE_NODE = 10 TRACE_MAIN_NODE = 11 TRACE_BOOT = 12 CRITICAL = 12 ERROR = 50 DATA_ERROR = 51 VIEW_ERROR = 52 DB_ERROR = 53 AUTH_ERROR = 54 NET_ERROR = 55 WARNING = 80 EVENT_BOOT = 81
data_index	integer	No	Indice en mensaje del agente
count	integer	No	Contador (reservado)
Tabla:	**lmp_logpath**	Flujo de operación logKey origen -> logKey destino	
Nombre	**Tipo**	**Nulos**	**Descripción**
id	integer	Si	Llave primaria

keymaster	text	Si	Llave o código único en el maestro
logkey1_id	integer	Si	id a logKey con el nodo origen
logkey2_id	integer	Si	id a logKey con el nodo destino
component_id	integer	Si	Id del componente
data_index	integer	No	Indice en mensaje del agente
samples_count	integer	No	Numero de muestras
duration_avg	real	No	Duración promedio de referencia
duration_std	real	No	desciación estandar de referencia
duration_max	real	No	Duración máxima de referencia
samples_start	timestamp without time zone	No	Fecha inicial de datos de referencia
samples_end	timestamp without time zone	No	Fecha final de datos de referencia
Tabla:	**lmp_remotecall**	Llamados entre componentes.	
Nombre	**Tipo**	**Nulos**	**Descripción**
id	integer	Si	Llave primaria
logkey1_id	integer	Si	id a logKey con el nodo origen
logkey2_id	integer	Si	id a logKey con el nodo destino
Tabla:	**lmp_logrecord**	Logrecords capturados para realizar el analisis de llamados remotos	
Nombre	**Tipo**	**Nulos**	**Descripción**
id	integer	Si	Llave primaria
exectime	timestamp without time zone	Si	Fecha y hora de ejecucion
logkey_id	integer	Si	Id LogKey
component_id	integer	Si	Id del componente
remotecallkey	text	No	Identificador que relaciona los mensajes entre nodos
userkey	text	No	Identificador que relaciona los usuarios
detail	text	No	Detalle adicional
Tabla:	**lmp_path_measure**	Medidas de flujos de operación por ventana de tiempo	
Nombre	**Tipo**	**Nulos**	**Descripción**
id	integer	Si	Llave primaria
date	timestamp without time zone	Si	Fecha y hora
period	integer	Si	Numero de periodo de tiempo (Reservado)
path_id	integer	Si	Id del logPath

host_id	integer	Si	Id del host
count	integer	No	Numero de muestras
duration_avg	real	No	Duración promedio
duration_std	real	No	Desviación estandar
duration_max	real	No	Duración máxima
performance	real	No	Indicador de rendimiento
ref	boolean	No	Indica que es un dato que sirve de referencia
performance_min	real	No	Indicador de rendimiento mínimo
Tabla:	**lmp_host_measure**	Datos del host medidos en una ventana de tiempo	
Nombre	**Tipo**	**Nulos**	**Descripción**
id	integer	Si	Llave primaria
date	timestamp without time zone	Si	Fecha y hora
period	integer	Si	Numero de periodo de tiempo (Reservado)
host_id	integer	Si	Id del host
source_id	integer	Si	Id de la fuente de datos
cpu	real	No	Porcentaje de uso de CPU
cpu_user	real	No	Porcentaje de uso de CPU de usuario
cpu_sys	real	No	Porcentaje de uso de CPU de sistema
cpu_idle	real	No	Porcentaje de CPU disponible
mem	real	No	% de utilización de memoria
swap	real	No	% de utilización de memoria virtual o SWAP
diskusage	real	No	% de utilizacion de disco duro
pids	real	No	Numero de procesos activos
cnxs	real	No	Numero de conexiones de red establecidas
users	real	No	Numero de usuarios del sistema operativo con sesion activa
disk_io_rate_w	real	No	Rata de escritura de disco duro
disk_io_rate_r	real	No	Rata de lectura de disco duro
net_io_rate_in	real	No	Rata de datos recibidos en las interfaces de red
net_io_rate_out	real	No	Rata de datos transmitidos en las interfaces de red
openfiles	real	No	Número de archivos abiertos
openfiles_rate	real	No	rata de cambio de archivos abiertos
net_err_rate_in	real	No	Contador de errores de red en transmisión

net_err_rate_out	real	No	Contador de errores de red en recepción
net_drop_rate_in	real	No	Contador de paquetes descartados en transmisión
net_drop_rate_out	real	No	Contador de paquetes descartados en recepción
ref	boolean	No	Indica que es un dato que sirve de referencia
Tabla:	**lmp_measure**	Datos genericos capturados en una ventana de tiempo	
Nombre	**Tipo**	**Nulos**	**Descripción**
id	integer	Si	Llave primaria
date	timestamp without time zone	No	Fecha y hora
period	integer	No	Numero de periodo de tiempo (Reservado)
type_id	integer	Si	Id del tipo de medida
source_id	integer	Si	Id de la fuente de datos
value	real	No	Valor
ref	boolean	No	Indica que es un dato que sirve de referencia
Tabla:	**lmp_result**	Resultados del rendimiento medido y estimado por componente	
Nombre	**Tipo**	**Nulos**	**Descripción**
id	integer	Si	Llave primaria
date	timestamp without time zone	Si	Fecha y hora
period	integer	Si	Numero de periodo de tiempo (Reservado)
host_id	integer	Si	Id del host
component_id	integer	No	Id del componente
performance	real	No	Indicador del rendimiento
predicted	real	No	Predicción del ndicador de rendimeinto
anomaly	integer	No	Predicción de anomalias
performance_min	real	No	Indicador del rendimiento mínimo
g1avg	real	No	Número de muestras con indicador de rendimiento promedio entre [0,75-1] (Reservado)
g2avg	real	No	Número de muestras con indicador de rendimiento promedio entre [0,5-0,75] (Reservado)
g3avg	real	No	Número de muestras con indicador de rendimiento promedio entre [0,25-0,5] (Reservado)
g4avg	real	No	Número de muestras con indicador de rendimiento

			promedio entre [0-0,25] (Reservado)
g1min	real	No	Número de muestras con indicador de rendimiento mínimo entre [0,75-1] (Reservado)
g2min	real	No	Número de muestras con indicador de rendimiento mínimo entre [0,5-0,75] (Reservado)
g3min	real	No	Número de muestras con indicador de rendimiento mínimo entre [0,25-0,5] (Reservado)
g4min	real	No	Número de muestras con indicador de rendimiento mínimo entre [0-0,25] (Reservado)
Tabla:	**lmp_sourcecounters**	Metricas del procesamiento de la herrameinta	
Nombre	**Tipo**	**Nulos**	**Descripción**
id	integer	Si	Llave primaria
date	timestamp without time zone	Si	Fecha y hora
source_id	integer	Si	Id de la fuente de datos
count	integer	No	Numero de lineas o muestras procesadas
bytes	integer	No	Número de bytes procesados
records	integer	No	Número de registros creados
fails	integer	No	Número de fallas

8. Bibliografía

[1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, vol. 2nd. Addison-Wesley, 2012.

[2] L. Chung and J. D. P. Leite, “On Non-Functional Requirements in Software Engineering,” *Concept. Model. Found. …*, pp. 363–379, 2009.

[3] M. Woodside, G. Franks, and D. C. Petriu, “The Future of Software Performance Engineering,” in *Future of Software Engineering (FOSE ’07)*, 2007, pp. 171–187.

[4] A. Van Hoorn, L. Grunske, D. Okanovic, and T. Pitakrat, “Hora : Architecture-aware online failure prediction,” *J. Syst. Softw.*, vol. 137, pp. 669–685, 2018.

[5] K. Nagaraj, C. Killian, and J. Neville, “Structured comparative analysis of systems logs to diagnose performance problems,” in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 353–366.

[6] D. Gunter, B. Tierney, B. Crowley, M. Holding, and J. Lee, “NetLogger: A toolkit for distributed system performance analysis,” in *IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems - Proceedings*, 2000, pp. 267–273.

[7] B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, and D. Gunter, “The NetLogger Methodology for High Performance Distributed Systems Performance Analysis,” in *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, 1998, p. 260--.

[8] J. Abeja and T. Debeaupuis, “Universal Format for Logger Messages,” 1999. [Online]. Available: https://tools.ietf.org/html/draft-abela-ulm-05.

[9] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, “Performance debugging for distributed systems of black boxes,” *ACM SIGOPS Operating Systems Review*, 2003. [Online]. Available: https://pdos.csail.mit.edu/~athicha/papers/blackboxes:sosp03.pdf.

[10] “Dot.” [Online]. Available: https://en.wikipedia.org/wiki/DOT_(graph_description_language).

[11] P. Reynolds and J. Wiener, “WAP5: black-box performance debugging for wide-area systems,” in *Proceedings of the 15th international conference on World Wide Web, WWW 2006*, 2006, pp. 1–10.

[12] Q. Fu, J. G. Lou, Y. Wang, and J. Li, “Execution anomaly detection in distributed systems through unstructured log analysis,” *Proc. - IEEE Int. Conf. Data Mining, ICDM*, pp. 149–158, 2009.

[13] L. Mariani and M. Pezzè, “Dynamic Detection of COTS Component Incompatibility,” *IEEE Softw.*, vol. 24, no. 5, pp. 76–85, Sep. 2007.

[14] K. Bare *et al.*, “ASDF: An automated, online framework for diagnosing performance problems,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6420 LNCS, pp. 201–226, 2010.

[15] H. Mi *et al.*, “Magnifier: Online Detection of Performance Problems in Large-Scale Cloud Computing Systems,” *2011 IEEE Int. Conf. Serv. Comput.*, pp. 418–425, 2011.

[16] S. Alspaugh, B. Chen, J. Lin, A. Ganapathi, M. Hearst, and R. Katz, “Analyzing Log Analysis: An Empirical Study of User Log Mining,” *28th Large Install. Syst. Adm. Conf.*, pp. 62–77, 2014.

[17] “Splunk.” [Online]. Available: https://www.splunk.com.

[18] S. S. and S. Z. Ledion Bitincka, Archana Ganapathi, “Optimizing Data Analysis with a Semi-structured Time Series Database,” *USENIX Assoc.*, pp. 7–7, 2010.

[19] C. A. Lai, J. Kimball, T. Zhu, Q. Wang, and C. Pu, “MilliScope: A Fine-Grained Monitoring Framework for Performance Debugging of n-Tier Web Services,” *Proc. - Int. Conf. Distrib. Comput. Syst.*, pp. 92–102, 2017.

[20] G. Coulouris, J. Dollimore, T. Kindberg, and B. Gordon, *Distributed Systems. Concepts and design*, 5th ed. Addison-Wesley, 2012.

[21] I. Foster, Y. Zhao, I. Raicu, and S. Lu, “Cloud Computing and Grid Computing 360-Degree Compared,” in *2008 Grid Computing Environments Workshop*, 2008, pp. 1–10.

[22] M. Fowler and J. Lewis, “Microservices,” 2014. [Online]. Available: http://martinfowler.com/articles/microservices.html.

[23] S. Newman, *Building Microservices*. O’Reilly Media, 2015.

[24] E. Zio and G. Sansavini, “Modeling interdependent network systems for identifying cascade-safe operating margins,” *IEEE Trans. Reliab.*, vol. 60, no. 1, pp. 94–101, 2011.

[25] M. A. Latib, S. A. Ismail, H. M. Sarkan, and R. C. Mohd Yusoff, “Analyzing log in big data environment: A review,” *ARPN J. Eng. Appl. Sci.*, vol. 10, no. 23, pp. 17777–17784, 2015.

[26] A. Oliner, A. Ganapathi, and W. Xu, “Advances and challenges in log analysis,” *Commun. ACM*, vol. 55, no. 2, pp. 55–61, 2012.

[27] D. Tovarnak, A. Vaseekova, S. Novak, and T. Pitner, “Structured and interoperable

logging for the cloud computing Era: The pitfalls and benefits," in *Proceedings - 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC 2013*, 2013, pp. 91–98.

[28] A. C. Müller and S. Guido, *Introduction to Machine Learning with Python*. O'Reilly Media, 2016.

[29] M. Swamynathan, *Mastering Machine Learning with Python in Six Steps*. Apress Media, 2017.

[30] Z. Li, A. Das, and J. Zhou, "Model Generalization and Its Implications on Intrusion Detection," in *3rd International Conference on Applied Cryptography and Network Security ACNS 2005*, 2005, pp. 222–237.

[31] A. Radosavljevic and R. P. Anderson, "Making better Maxent models of species distributions: Complexity, overfitting and evaluation," *J. Biogeogr.*, vol. 41, no. 4, pp. 629–643, 2014.

[32] A. F. S. Prisco and M. J. Freddy Duitama, "Intrusion detection system for SCADA platforms through machine learning algorithms," *2017 IEEE Colomb. Conf. Commun. Comput. COLCOM 2017 - Proc.*, pp. 1–6, 2017.

[33] A. Andriani, "Application of C4.5 algorithm for detection of cooperatives failure in province level," in *International Seminar on Scientific Issues and Trends (ISSIT)*, 2014.

[34] "Sistema para la Detección de Intrusos en Plataformas SCADA Andrés Felipe Sánchez Prisco Universidad de Antioquia," pp. 1–104, 2017.

[35] J. A. Baena, "LogMapper HomePage," 2018. [Online]. Available: http://www.logmapper.org.

[36] SciPy developers, "scipy," 2018. [Online]. Available: https://scipy.org/.

[37] INRIA and others., "scikit-learn." [Online]. Available: http://scikit-learn.org/.

[38] NetworkX developers, "NetworkX." [Online]. Available: https://networkx.github.io/.

[39] O. I. C. S.A.S., "OSP International CALA S.A.S.," 2018. [Online]. Available: http://www.ospinternational.com/.

[40] Pivotal, "Spring Framework." [Online]. Available: https://spring.io/.

[41] "No Title." [Online]. Available: https://www.postgresql.org/.

[42] "Primefaces." [Online]. Available: https://www.primefaces.org/.

[43] Google, "Regions and Zones," 2018. [Online]. Available: https://cloud.google.com/compute/docs/regions-zones/.

[44] Apache Software Foundation, "Apache JMeter™." [Online]. Available: https://jmeter.apache.org/.

USO EXCLUSIVO DEL PROGRAMA

Acta comité _____ **Fecha**_____

Favor tener en cuenta que los evaluadores propuestos no deben pertenecer a su mismo grupo de investigación, deben tener un título igual o superior al cual aspira el estudiante de posgrado y experiencia en investigación acreditada, vinculados a Universidades u organismos de enseñanza superior o investigación. No deben tener relación con la propuesta a evaluar ni publicaciones recientes con el estudiante, tutor o grupo de investigación acerca del tema de la propuesta, trabajo o tesis.

Para la selección de evaluadores se tendrá lo estipulado en el reglamento de posgrados de la Facultad de Ingeniería:

Para propuesta o trabajo de Maestría

- Serán 2 jurados con título de Magister o Doctor
- Por lo menos uno de los evaluadores debe ser externo a la Universidad de Antioquia
- Uno de ellos o ambos pueden ser internacionales

Para las propuestas de investigación de Doctorado

- Serán 2 jurados con título de Doctor
- Ambos jurados deben ser externos a la Universidad de Antioquia
- Por lo menos uno de los jurados debe ser internacional

Para las tesis Doctorales

- Serán 3 jurados con título de Doctor
- Por lo menos 2 de los jurados deben ser externos a la Universidad de Antioquia
- Por lo menos uno de los jurados debe ser internacional

Nombre y Firma del estudiante
C.C. 71781798

Nombre y Firma del Director
C.C. 71598507