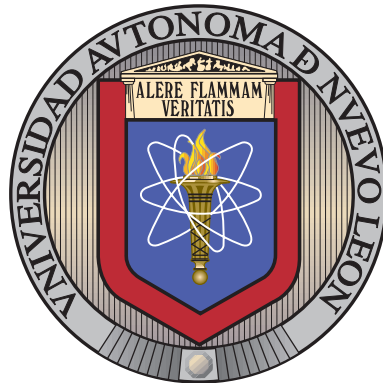


UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

SUBDIRECCIÓN DE ESTUDIOS DE POSGRADO



DESCUBRIMIENTO DE RELACIONES ENTRE
MICROSERVICIOS A TRAVÉS DE UN GRAFO
RELAJADO DE DEPENDENCIAS EXTRAÍDO CON
ANÁLISIS DE BITÁCORAS

POR

JAVIER IRACHETA VILLARREAL

COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE

DOCTORADO EN INGENIERÍA

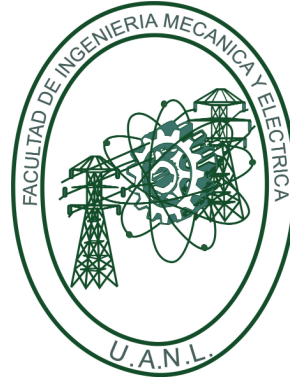
ORIENTACIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

AGOSTO 2024

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

SUBDIRECCIÓN DE ESTUDIOS DE POSGRADO



DESCUBRIMIENTO DE RELACIONES ENTRE
MICROSERVICIOS A TRAVÉS DE UN GRAFO
RELAJADO DE DEPENDENCIAS EXTRAÍDO CON
ANÁLISIS DE BITÁCORAS

POR

JAVIER IRACHETA VILLARREAL

COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE

DOCTORADO EN INGENIERÍA

ORIENTACIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

AGOSTO 2024

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
Facultad de Ingeniería Mecánica y Eléctrica
Posgrado

Los miembros del Comité de Evaluación de Tesis recomendamos que la Tesis "Descubrimiento de relaciones entre microsistemas a través de un grafo relajado de dependencias extraído con análisis de bitácoras", realizada por el estudiante Javier Iracheta Villarreal, con número de matrícula 1034020, sea aceptada para su defensa como requisito parcial para obtener el grado de Doctorado en Ingeniería con orientación en Tecnologías de la información.

El Comité de Evaluación de Tesis

Dra. Sara Elena Garza Villarreal
Director

Dr. César Emilio Villarreal
Revisor

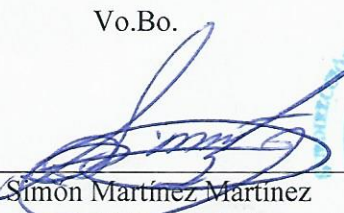
Dra. Leticia Amalia Neira Tovar
Revisor

Dra. María Teresa Pérez Morales
Revisor

Dr. Frumencio Olivas Álvarez
Revisor

Dr. César Guerra Torres
Revisor

Vo.Bo.



Dr. Simón Martínez Martínez
Subdirector de Estudios de Posgrado



Institución 190001

Programa 557525

Acta Núm. 352

Ciudad Universitaria, a 11 de septiembre de 2024.

A mi ignorancia que hace me sienta asombrado y humilde por cada nuevo descubrimiento.

A mi Asesora, la Dra Sara Garza, quien fue una guía definitiva para cruzar este camino llamado programa doctoral.

A mis padres que me inculcaron el hambre por aprender.

Finalmente a mi esposa e hijas, quienes llevaron conmigo las largas horas para completar este trabajo incluyendo sus alegrías y frustraciones.

ÍNDICE GENERAL

Resumen	xii
1. Introducción	1
1.1. Motivación y justificación	1
1.1.1. Arquitecturas basadas en servicios	2
1.1.2. La dificultad en las pruebas para las arquitecturas SOA	3
1.1.3. Importancia de las relaciones entre microservicios para generar pruebas	4
1.2. Definición del problema	6
1.2.1. Alcance	8
1.3. Protocolo	8
1.3.1. Hipótesis	8
1.3.2. Preguntas de investigación	8
1.3.3. Objetivos	9
1.4. Modelo de solución	9
1.5. Contribuciones	10

1.6. Organización del documento	10
2. Marco Teórico	12
2.1. Microservicios	12
2.2. Teoría de grafos	16
2.2.1. Redes complejas	17
2.3. Minería de procesos	19
2.3.1. Definiciones y notación	22
2.4. Resumen	25
3. Estado del Arte	26
3.1. Minería de procesos	26
3.2. Mejoramiento de arquitecturas basadas en microservicios	30
3.3. Descubrimiento de relaciones	34
3.4. Grafos aplicados al desarrollo de software	36
3.5. Resumen	40
4. Solución propuesta	44
4.1. Descripción general del modelo de solución	45
4.2. Bitácoras simples de microservicios	46
4.3. Procesamiento de una bitácora simple de microservicios	49
4.3.1. Procesamiento lógico	49

4.3.2. Procesamiento físico	50
4.4. Descubrimiento de relaciones entre microservicios	53
4.4.1. Grafo de secuencia relajado	53
4.4.2. Grafo de dependencia relajado	57
4.5. Resumen del capítulo	57
5. Experimentos y Resultados	58
5.1. Preparación de los datos	58
5.1.1. Extracción	59
5.1.2. Refinamiento	64
5.2. Pruebas de cobertura	68
5.2.1. Configuración experimental	70
5.2.2. Resultados y Discusión	76
5.3. Pruebas de descubrimiento	80
5.3.1. Configuración experimental	80
5.3.2. Resultados y Discusión	82
5.4. Resumen	84
6. Conclusiones y Trabajo futuro	86
6.1. Resumen	86
6.2. Respuesta a las preguntas de investigación	87
6.3. Contribuciones	88

6.4. Posibles aplicaciones	89
6.5. Trabajo futuro	90
6.6. Comentarios finales	91
A. Anexos	92

ÍNDICE DE FIGURAS

2.1. Línea de tiempo de la tecnología de microservicios	13
2.2. Ejemplo de una bitácora de servicios	16
2.3. Flujo de la minería de procesos	20
2.4. Representación del flujo de datos y procesos	21
2.5. Estructura de un registro de eventos.	23
3.1. Principales algoritmos de minería	27
4.1. Ejemplo del cálculo de $w(a, b)$ con $h = 4$ (Eq. 4.3)	56
5.1. Ejemplo de KQL	61
5.2. Detalle de las consultas	63
5.3. Ejemplo de actividades (únicas) registradas	66
5.4. Ejemplo de vector en la bitácora refinada	66
5.5. Descripción de la frecuencia de registros (escala logarítmica)	67
5.6. Descripción de la frecuencia de registros	69
5.7. Herramienta de monitoreo de redes	73

5.8. Código de python	73
5.9. Resultados de exhaustividad	76
5.10. Resultados de exhaustividad con diferentes niveles de h	77
5.11. Medias de muestras con niveles de $h = 1$ hasta $h = 5$	82
5.12. Resultados de descubrimiento	84
A.1. Anexo con detalles de los resultados	92

ÍNDICE DE TABLAS

3.1. Trabajos relacionados con minería de procesos	41
3.2. Mejoramiento de arquitecturas	42
3.3. Grafos aplicados al desarrollo de software	43
4.1. Ejemplo de una bitácora concurrente	48
5.1. Resumen de la extracción	59
5.2. Propiedades de los grafos de dependencia generados	70
5.3. Características de los flujos de negocio obtenidos	74
5.4. Resumen de los resultados de exhaustividad	78
5.5. Resumen de los resultados de la validación	83

RESUMEN

Javier Iracheta Villarreal.

Candidato para obtener el grado de Doctorado en Ingeniería Orientación en Tecnologías de la Información.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Título del estudio: DESCUBRIMIENTO DE RELACIONES ENTRE MICROSERVICIOS A TRAVÉS DE UN GRAFO RELAJADO DE DEPENDENCIAS EXTRAÍDO CON ANÁLISIS DE BITÁCORAS.

Número de páginas: 101.

OBJETIVOS Y MÉTODO DE ESTUDIO

OBJETIVO PRINCIPAL. *Generar un grafo de dependencia relajado de microservicios a partir de una bitácora simple de producción, donde esta bitácora es restringida, masiva y concurrente.*

OBJETIVOS SECUNDARIOS

- Definir un método para procesar la bitácora de microservicios que permita una manipulación repetible.
- Definir un método para crear un conjunto de datos que sirvan como base para

ejecutar las mediciones de precisión de los diferentes grafos resultantes de la experimentación.

MÉTODO DE ESTUDIO. El modelo propuesto se basa en las medidas de secuencia y dependencia generadas por el *minero heurístico* (Weijters y Ribeiro, 2011), las cuales han sido extendidas (“relajadas”) para crear un grafo de dependencia. Para relajar la definición de medida de secuencia, adaptamos el concepto de la teoría de grafos conocido como “vecinos de i saltos”, donde un vecino de 0 saltos es el nodo en sí, un vecino de 1 salto es el vecino directo de un nodo, un vecino de 2 saltos es el vecino del vecino del nodo, y así sucesivamente. En nuestro contexto, un evento consiste en una llamada a microservicio (es decir, se invoca una dirección a través de algún protocolo de red). De esta manera, para un evento e , un evento de 0 saltos es este mismo e , mientras que un evento de 1 salto sería un evento f inmediatamente anterior a e .

CONTRIBUCIONES Y CONCLUSIONES. En resumen, nuestras aportaciones son las siguientes:

Enfoque sin identificador de traza. El minero heurístico considera los tres componentes de descubrimiento del proceso: identificador de traza, evento y firma de tiempo. El método propuesto se utiliza únicamente para evento y firma de tiempo. El estado del arte para casos similares requiere estructuras adicionales a la bitácora.

Enfoque de meta-traza. Se trata todo el registro de eventos entero como un solo proceso, ya que no hay identificadores de traza. Este enfoque está respaldado por datos masivos y patrones de repetición esperados.

Enfoque relajado. Implementando una contribución degradada para el grafo de secuencia basado en la distancia del evento en los registros y el conteo de frecuencias. La contribución degradada es producto de un nivel de relajación h incluido en las medidas de secuencia y dependencia.



La Sociedad Mexicana de Ciencias de la Computación y la Universidad Autónoma de Coahuila a través del Centro de Investigación en Matemáticas Aplicadas y la Facultad de Sistemas

otorgan la presente

CONSTANCIA

a

Javier Iracheta y Sara E. Garza

Por su destacada participación en el Coloquio de estudiantes con el tema "Generación de un grafo probabilístico en base a bitácoras de microservicios" en el Encuentro Nacional en Computación 2020 efectuado del 24 al 26 de agosto de 2020

Dra. María Lucía Barrón Estrada
Presidenta de la SMCC

Dra. Valeria Soto Mendoza
Organizador General Local



Valida el reconocimiento en nuestro sitio escaneando el QR.
O ingresa en <https://enc-2020.web.app/validacion/ENC-0197>

Figura 1: Certificado de participación

PUBLICACIONES. Después de la fase de procesamiento inicial de la bitácora de eventos, se diseñó e implementó una metodología para procesar y validar inicialmente la hipótesis propuesta, estos avances fueron documentados y publicados en la monografía *Aplicaciones de la computación* (Iracheta y Garza, 2020), la Figura 1 muestra la constancia de dicho evento.

Firma de la asesora: Sara E. Garza Villarreal
Dra. Sara E. Garza Villarreal

CAPÍTULO 1

INTRODUCCIÓN

En el ámbito de la tecnología actual, se han vuelto críticos tanto el diseño eficiente como la gestión efectiva de los sistemas distribuidos. La arquitectura basada en servicios web ha surgido como una respuesta ante esta problemática, atendiendo los aspectos de escalabilidad y mantenimiento al dividir las aplicaciones en pequeños componentes independientes llamados *servicios*. Sin embargo, la complejidad inherente a estas arquitecturas plantea nuevos retos, especialmente en la identificación y comprensión de las relaciones entre estos servicios. En la presente investigación, se aborda el descubrimiento automático de relaciones entre servicios (específicamente, entre un sub-tipo denominado *microservicios*), para lo cual se construye un grafo de dependencia relajado a partir de la información contenida en las bitácoras del sistema.

1.1 MOTIVACIÓN Y JUSTIFICACIÓN

En el desarrollo de software, las pruebas de integración evalúan —a través de simulaciones— el manejo de errores y fallos entre los diferentes elementos de un sistema, lo cual es crucial para garantizar que este, en su conjunto, sea resiliente. En una arquitectura basada en servicios, donde estos generalmente interactúan entre sí y con otros componentes del sistema, las pruebas de integración garantizan que estas

interacciones funcionen correctamente y que los servicios se comporten de la manera esperada cuando se conectan entre sí. Viendo que estas interacciones o *relaciones* entre servicios son altamente relevantes, una estrategia que permita identificar dichas relaciones podría ayudar de manera significativa para armar escenarios de prueba en este tipo de arquitectura.

1.1.1 ARQUITECTURAS BASADAS EN SERVICIOS

En las arquitecturas monolíticas, los procesos están asociados y se ejecutan como un solo servicio, lo que simplifica su desarrollo pero dificulta la escalabilidad y la implementación de cambios. Este modelo arquitectónico aumenta en complejidad al agregar o mejorar características, limita la experimentación y eleva el riesgo de fallas que afectan la disponibilidad de la aplicación debido a la alta dependencia entre procesos (Kuryazov *et al.*, 2020).

Como respuesta a estos problemas, surge la arquitectura orientada a servicios (Mankovski, 2009), que en lo subsecuente denominaremos *SOA*, por *Service Oriented Architecture*. Esta arquitectura crea aplicaciones con componentes independientes que ejecutan los procesos como servicios web, y que se comunican mediante interfaces bien definidas. Al ser independientes, estos servicios pueden actualizarse, implementarse y escalarse según las demandas específicas de la aplicación.

La evolución y madurez de la arquitectura *SOA* ha permitido que la adopción se esté incrementando con velocidad en múltiples áreas —Esto incluye integración en sistemas empresariales, redes sociales y aplicaciones de dispositivos móviles para el usuario. El componente principal que hace que *SOA* tenga un alto nivel de adopción es precisamente su capacidad para integrar aplicaciones en diferentes dominios. Esto es particularmente relevante porque significa que una red de servicios puede tener alcance fuera de la organización. Tengamos como ejemplo los servicios expuestos por grandes compañías como *Google* y *Amazon*.

El que un servicio web esté *expuesto* significa que el proveedor de tal servicio no tiene control de cuántos ni de quiénes lo están usando en un punto particular del tiempo. Es decir, si bien pueden implementar mecanismos de registro y control de acceso, no tienen control sobre el impacto que un cambio en un servicio determinado pueda causar a los sistemas del consumidor de ese servicio. Este punto es muy relevante para destacar el análisis de dependencia de un servicio web en una determinada red.

1.1.2 LA DIFICULTAD EN LAS PRUEBAS PARA LAS ARQUITECTURAS SOA

Debido a que el desarrollo de software en actividades empresariales y comerciales demanda flexibilidad para integrar recursos internos y externos (Gouigoux y Tamzalit, 2017), la arquitectura *SOA* ha demostrado ser eficiente en aplicaciones basadas en servicios web (Pautasso *et al.*, 2017). Sin embargo, a pesar de sus beneficios, también ha creado nuevos retos. Uno de ellos reside en las pruebas, ya que estas presentan dificultades particulares debido a que los servicios suelen verse como componentes desde la perspectiva del consumidor (Mao, 2009).

Una de las principales características de un servicio web es su encapsulamiento, es decir, que tiene un contrato que se usa como entrada donde se define exactamente qué datos necesita para ser ejecutado, y de manera análoga, tiene el mismo mecanismo de salida, es decir, un consumidor de dicho servicio usa un conjunto de datos específicos de entrada y espera una salida definida. Este fenómeno de la separación y encapsulamiento de código hace que sea difícil probar servicios web más allá de una prueba unitaria sin tener conocimiento detallado de todos los servicios que componen una arquitectura en particular. Por lo tanto, probar un determinado servicio web típicamente significa tomar sólo en cuenta los parámetros requeridos para una invocación y la evaluación de la respuesta específica; también es común usar for-

matos como XML o JSON utilizando la especificación del servicio como método de prueba. Otras pruebas incluyen validaciones de disponibilidad (Mankovski, 2009).

Los servicios evolucionan rápidamente, y esto representa cambios, donde dichos cambios pueden impactar la estabilidad operativa de los sistemas que los consumen. Tomando en cuenta lo anterior, cuando hay un cambio en un servicio, no se puede garantizar la nulidad de impactos. Por lo tanto, la mayoría de las veces—para mitigar estos impactos—se procede a crear una nueva versión del mismo servicio, de modo tal que en un tiempo dado pueden coexistir múltiples versiones del mismo servicio usadas por diferentes clientes. Debido a esto, las aplicaciones basadas en servicios presentan aún más desafíos.

Las pruebas de servicios también se ven afectadas por la falta de documentación detallada sobre la interacción y semántica de los datos intercambiados (Nguyen *et al.*, 2011). Este contexto agrava problemas comunes, como la selección y priorización de escenarios de prueba. La evaluación de los resultados esperados de las pruebas tampoco es sencilla, ya que la información proporcionada por los lenguajes de descripción de interfaz es limitada. Se ve igualmente limitada la capacidad de prueba de clientes externos que consumen los servicios, que además involucra costos asociados significativos. Esto impone restricciones en la cantidad y calidad de pruebas que pueden hacerse, en comparación con el software tradicional.

1.1.3 IMPORTANCIA DE LAS RELACIONES ENTRE MICROSERVICIOS PARA GENERAR PRUEBAS

Una evolución de los servicios web tradicionales son los *microservicios*. Estos son un enfoque compuesto por pequeños servicios independientes que representan funcionalidades atómicas y que se comunican a través de una *API* (Interfaz de Programación de Aplicaciones) bien definida. Las arquitecturas de microservicios, al estar fragmentadas en funcionalidades más pequeñas que las arquitecturas basadas

en servicios web, hacen que las aplicaciones sean más fáciles de escalar y más rápidas de desarrollar y los cambios son mucho más focalizados y rápidos. Esto permite la innovación y acelera el tiempo de comercialización de las nuevas características.

Una de las ventajas de esta evolución de los servicios web en microservicios es el registro de actividad en bitácoras especializadas y destinadas para su análisis. Estas bitácoras registran eventos para rastrear e informar los datos de la aplicación de manera centralizada. Los eventos registrados proporcionan una descripción general del estado de ejecución de la aplicación, realizan un seguimiento de los errores de código o fallas de la aplicación y envían mensajes informativos. Siempre que una aplicación falla, con el análisis de bitácoras es posible la mayor parte del tiempo, determinar la potencial causa de falla y el momento de la misma.

Entender la relación entre servicios es esencial para desarrollar estrategias de prueba efectivas a nivel de servicio y para comprender mejor el impacto de los cambios en servicios específicos. Las bitácoras de servicios, que registran cada llamada de servicio con algunos atributos como lo son su firma de tiempo, son valiosos para esta tarea. Estos datos se pueden tratar utilizando la minería de procesos, especialmente para descubrir y comprender la secuencia y dependencia entre los servicios.

Viendo lo anterior, la motivación principal de esta investigación radica en la necesidad de desarrollar un método que descubra la relación inherente entre servicios. Para ello, proponemos la creación de un grafo, donde cada nodo representa un servicio y cada conexión indica la dirección y fuerza de la relación entre un par de servicios. La comprensión intuitiva en términos de la complejidad de una arquitectura dada, es una ventaja de entender estas relaciones entre servicios.

Utilizando bitácoras de eventos como entrada, en el presente trabajo de investigación se ha desarrollado una adaptación del *minero heurístico* (Weijters y Ribeiro, 2011). Esta adaptación utiliza una medida de secuencia *relajada* y un grafo de dependencia (también relajado) para mejorar la identificación de relaciones entre las llamadas a microservicios registradas en una bitácora—incluso en situaciones don-

de hay registros de eventos sin un identificador que ayude a agrupar los servicios por el proceso que los invocó. Nuestro método propuesto considera toda la bitácora de eventos como un único proceso, proporcionando una visión más clara de las relaciones y dependencias entre los servicios.

1.2 DEFINICIÓN DEL PROBLEMA

En el presente trabajo, el problema a abordar consiste en el *descubrimiento de relaciones entre microservicios*. Para este problema, se tiene como entrada una bitácora restringida y se produce como salida un grafo dirigido y ponderado donde cada nodo representa un microservicio y cada conexión representa una relación de precedencia entre un par de servicios. Debido a que las relaciones de precedencia entre servicios no serían necesariamente inmediatas, consideramos la generación de un *grafo de dependencia relajado*. Asimismo, se considera que una bitácora restringida solamente cuenta con un identificador de actividad y una firma de tiempo, donde el identificador corresponde a la invocación de un servicio.

CONTEXTO. Como se mencionó anteriormente, este problema se desarrolla en el ámbito del uso empresarial de microservicios. En este ámbito, existe una cantidad considerable de secuencias de invocación de dichos servicios, ya que estas secuencias corresponden a diferentes aplicaciones; es decir, cada empresa cuenta con diferentes requerimientos de acuerdo a su giro y, por consiguiente, puede ensamblar los microservicios disponibles de distintas maneras. Lo anterior limita la generación de escenarios de prueba por parte de los equipos de *testing* proveedores de los servicios y crea la necesidad de encontrar alternativas para complementar los escenarios existentes. En ese sentido, el descubrimiento de relaciones entre microservicios facilitaría una generación más amplia de escenarios de prueba.

SUPOSICIONES Y RESTRICCIONES. La principal suposición en este trabajo es que, si una bitácora restringida es masiva, va a tender a exhibir secuencias frecuentes de invocación (i.e., patrones). Por lo tanto, también se supone que las relaciones espurias serán menos frecuentes (i.e., el ruido—por ejemplo, a causa de las concurrencias). De igual forma, se supone que la bitácora a utilizar será masiva, restringida y con concurrencias. También se supone que no se cuenta con estructuras auxiliares que indiquen cuáles son los procesos que dan origen a la bitácora (mismas que son utilizadas en otros trabajos); de igual manera, se supone que el descubrimiento de los procesos no es necesario, sino solamente el descubrimiento de relaciones. Es decir, lo que nos interesa es descubrir nexos entre diferentes microservicios (sobre todo aquellos que pudieran ser difíciles de detectar por los probadores), y no tanto los procesos de las empresas que utilizan estos microservicios, ya que—como se expuso previamente—se tiene una gran cantidad de procesos donde pudieran ser llamados los servicios.

PARÁMETROS (VARIABLES DE ESTUDIO). La variable que se va a medir es la correspondencia entre el grafo de dependencia relajado y diferentes grafos de referencia (tipo *ground truth*); como veremos más adelante, esta correspondencia estará dada por la métrica de exhaustividad (*recall*). Otra variable que se va a medir es la probabilidad de que las relaciones descubiertas (es decir, aquellas que no se encuentren en los grafos de referencia) sean significativas—es decir, que no representen ruido. Para ello, se utilizarán modelos de grafos aleatorios.

SALIDA ESPERADA. La salida esperada sería un grafo relajado de dependencias, donde cada nodo representa un microservicio y cada conexión representa una relación de precedencia (no necesariamente inmediata) entre un par de microservicios. Este grafo sería dirigido y ponderado. La dirección en las conexiones representaría el orden de invocación y la ponderación representaría la intensidad con la que se presenta la relación de precedencia entre los servicios involucrados.

1.2.1 ALCANCE

El presente trabajo consiste en el procesamiento y análisis de una bitácora de microservicios para la generación de un grafo de dependencia, donde cada nodo representa un microservicio y cada conexión, una relación de precedencia entre un par de microservicios. A pesar de que el fin último de este grafo es la generación de escenarios de prueba, dicha generación no se considera dentro del alcance. Asimismo, solamente se considera trabajar con *bitácoras simples de microservicios*, es decir, con bitácoras que son restringidas, concurrentes y masivas. Otro tipo de bitácoras se dejan como trabajo futuro.

1.3 PROTOCOLO

A continuación, se describen la hipótesis, objetivos y preguntas de investigación.

1.3.1 HIPÓTESIS

Es posible generar un grafo de dependencia relajado a partir de una bitácora simple de microservicios.

1.3.2 PREGUNTAS DE INVESTIGACIÓN

- ¿Cómo se puede procesar una bitácora de microservicios, de tal manera que pueda extraerse un grafo que represente las relaciones entre microservicios?
- ¿Cómo se puede inferir la relación entre dos microservicios utilizando una bitácora simple?
- ¿Cómo se puede medir la calidad de los resultados?

1.3.3 OBJETIVOS

OBJETIVO PRINCIPAL. *Encontrar las relaciones entre microservicios generando un grafo de dependencia relajado a partir de una bitácora simple de microservicios, donde esta bitácora es restringida, masiva y concurrente.*

Objetivos secundarios:

- Definir un método para procesar la bitácora de microservicios que permita una manipulación repetible.
- Definir un método para crear un conjunto de datos que sirvan como base para ejecutar las mediciones de precisión de los diferentes grafos resultantes de la experimentación.

1.4 MODELO DE SOLUCIÓN

El modelo propuesto se basa en las medidas de secuencia y dependencia generadas por el *minero heurístico* (Weijters y Ribeiro, 2011), las cuales han sido extendidas (“relajadas”) para crear un grafo de dependencia. Para relajar la definición de medida de secuencia, adaptamos el concepto de la teoría de grafos conocido como “vecinos de i saltos”, donde un vecino de 0 saltos es el nodo en sí, un vecino de 1 salto es el vecino directo de un nodo, un vecino de 2 saltos es el vecino del vecino del nodo, y así sucesivamente. En nuestro contexto, un evento consiste en una llamada a microservicio. De esta manera, para un evento e , un evento de 0 saltos es este mismo e , mientras que un evento de 1 salto sería un evento f inmediatamente anterior a e . Las fases del modelo de solución se enuncian a continuación:

- Acondicionamiento de datos

- Creación de un grafo de secuencia relajado
- Creación de un grafo de dependencia relajado

Como parte del modelo de solución, están los métodos de validación de resultados. Se propone desarrollar dos métodos de validación: el primero se basa en el uso de relaciones conocidas que nos ayuden a validar las relaciones encontradas (uso de *ground truth*), y el segundo se basa en un método de validación de las relaciones encontradas contra las relaciones que se pudieran encontrar por azar (uso de grafos aleatorios).

1.5 CONTRIBUCIONES

Las contribuciones del presente trabajo consisten en:

1. Un método para interpretar y procesar una bitácora de registros que es masiva, restringida y concurrente
2. Un método para la generación de un grafo que represente los servicios y sus relaciones, donde este método incluye generar grafos de secuencia y dependencia relajados
3. Un método para la evaluación de resultados que sea repetible y utilizable para contextos similares
4. Evidencia de la calidad del grafo de dependencia generado

1.6 ORGANIZACIÓN DEL DOCUMENTO

La organización del presente escrito consta de los siguientes capítulos y objetivos: el Capítulo 2 define y explica los conceptos necesarios para tener el contexto de

la investigación, a saber: teoría de grafos, microservicios y minería de procesos. El Capítulo 3 expone las investigaciones relacionadas más relevantes, tales como descubrimientos de procesos con trazas sin etiquetas, y métodos diversos que actualmente se usan para llegar a un resultado similar al propuesto. Por otra parte, el Capítulo 4 define los pasos que llevan el flujo de la investigación y la coherencia entre ellos en términos de los resultados esperados. De igual manera, en el Capítulo 5 se describen los experimentos realizados, se analizan los resultados encontrados y se provee la discusión correspondiente. Por último, en el Capítulo 6 se reflexiona sobre los resultados obtenidos y se proponen líneas futuras de investigación.

CAPÍTULO 2

MARCO TEÓRICO

En este capítulo, se definen una serie de temas y conceptos base para la comprensión de capítulos subsecuentes. Entre estos temas se encuentran (a) los microservicios, que son punto central para esta investigación; (b) conceptos relevantes de teoría de grafos; y (c) conceptos relacionados con minería de procesos. Los conceptos de estos dos últimos temas coadyuvarán la comprensión de la solución propuesta para el descubrimiento automático de relaciones entre microservicios.

2.1 MICROSERVICIOS

Un *microservicio* es un servicio atómico, con una sola responsabilidad y con una funcionalidad única tal que se puede implementar, escalar y probar de forma independiente (Larrucea *et al.*, 2018). Al basarse en funcionalidades independientes con límites claros, los microservicios pertenecen a una *arquitectura orientada a servicios (SOA)*, de la que se habló brevemente en el Capítulo 1; por tanto, podría decirse que los microservicios son un tipo particular de *SOA* (Josuttis, 2007). Este tipo de servicio surgió como parte de la evolución de los servicios web, que fueron igualmente mencionados en el Capítulo 1. En la Figura 2.1, se puede apreciar esta evolución, así como el punto de aparición del concepto de *microservicio*, y también el cómo se fueron dando diferentes atributos a través del tiempo a medida que la

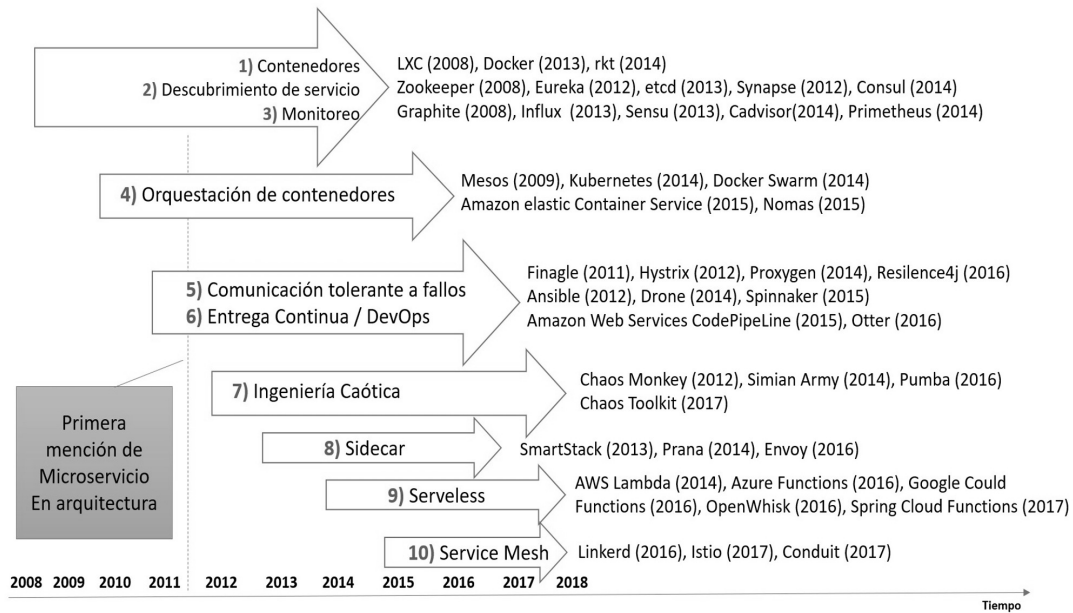


Figura 2.1: Línea de tiempo de la tecnología de microservicios

tecnología avanzaba y permitía implementar estos conceptos (Jamshidi *et al.*, 2018).

Algunos de los atributos relevantes en una arquitectura basada en microservicios son los siguientes:

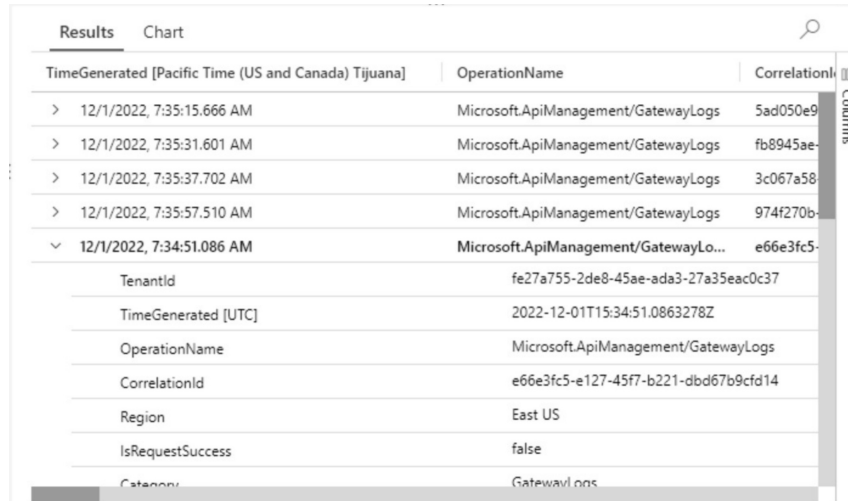
- *Componentización.* Un componente es un elemento de software que tiene la capacidad de ser reemplazado y mejorado de forma individual. Esta es la principal razón para utilizar servicios como componentes: los servicios se pueden implementar de forma independiente (Alshuqayran *et al.*, 2016).
- *Organización en torno a las capacidades empresariales.* La forma en que una organización se comunica internamente determina la estructura del sistema que diseña, de manera que los microservicios reflejan la organización a la que pertenecen (Conway, 1968).
- *Productos, no proyectos.* A diferencia de un enfoque tradicional de producción de software, donde se hace desarrollo y entrega, una arquitectura de microservicios (*MSA*) suele tener el enfoque de producto, es decir, está en constante desarrollo y operación (Francesco *et al.*, 2017).

- *Puntos finales inteligentes y tuberías tontas.* La complejidad del servicio está en el servicio mismo, y no en la forma de comunicación con otros servicios. Este atributo hace énfasis en utilizar la forma más simple y efectiva de comunicación entre servicios—ejemplo de ello es el uso de *HTTP*, en contraste con otros sistemas de comunicación compleja entre diferentes elementos de la arquitectura (Di Francesco, 2017).
- *Descentralización.* Este atributo permite que múltiples plataformas de desarrollo puedan coexistir, esto es, un servicio se puede desarrollar en un lenguaje “x” y otro en un lenguaje “y” mientras se publiquen interfaces que permitan consumirlos.
- *Descentralización en los datos.* Esta propiedad es que cada microservicio puede gestionar su propia base de datos, ya sea usando diferentes instancias con la misma tecnología de base de datos o diferentes sistemas de bases de datos. A este enfoque se le conoce como *persistencia políglota* (Kaur *et al.*, 2021).
- *Automatización de la infraestructura.* El avance de las técnicas para automatizar las infraestructuras ha sido enorme en los últimos tiempos. La evolución de la nube ha reducido la complejidad operativa de crear, implementar y operar microservicios, y esto permite automatizar el proceso desde la actualización de código hasta la instalación en un ambiente productivo (Railić y Savić, 2021).
- *Diseño para el fallo.* Los equipos de microservicios implementan configuraciones sofisticadas de monitoreo y registro para cada servicio individual, como paneles que muestren el estado arriba/abajo y una variedad de métricas operativas y comerciales relevantes (Cinque *et al.*, 2021).
- *Diseño evolutivo.* Una característica esencial de un microservicio es la posibilidad de ser reemplazado y actualizado de manera independiente. Esto implica que se busquen puntos en los que es posible reescribir un componente sin afectar a sus colaboradores (Assunção *et al.*, 2023).

Para lograr tener estos atributos, que son deseables en diversos contextos empresariales e industriales, los microservicios emplean mayormente un protocolo de comunicación denominado *REpresentational State Transfer (REST)* o *Transferencia de estado representacional*. Este protocolo proporciona los elementos necesarios para establecer una comunicación clara entre el proveedor del servicio y el consumidor de este. Otro protocolo comúnmente utilizado en el entorno de Internet es el de transferencia de hipertexto (*HTTP*), además de formatos de comunicación ligeros y nativos para el desarrollo web.

Los atributos mencionados a nivel de microservicio y de arquitectura de microservicios (*MSA*) nos dan una idea general de las capacidades de esta tecnología. A pesar de que existe gran detalle técnico para una comprensión profunda de este tema, para efectos del contexto de esta investigación, se han nombrado los principales puntos clave que motivan la generación de un grafo de dependencia para inferir las relaciones entre microservicios, y cómo este puede ayudar a mitigar problemas generados por los atributos evolutivos y de aislamiento intrínsecos en esta arquitectura.

Otro de los elementos claves que motiva el descubrimiento de relaciones en una arquitectura basada en microservicios es la disponibilidad de *bitácoras*, mismas que se usan como herramienta de monitoreo en estas arquitecturas. Debido a que la generación de bitácoras se practica frecuentemente a través de las diferentes compañías que construyen dichas arquitecturas, existe una base de información a la cual recurrir para realizar diferentes estudios—como es el caso de la presente investigación. La Figura 2.2 muestra un ejemplo de cómo luce la administración de una bitácora de servicios montados sobre *Microsoft AZURE*.



TimeGenerated [Pacific Time (US and Canada) Tijuana]	OperationName	CorrelationId
> 12/1/2022, 7:35:15.666 AM	Microsoft.ApiManagement/GatewayLogs	5ad050e9-
> 12/1/2022, 7:35:31.601 AM	Microsoft.ApiManagement/GatewayLogs	fb8945ae-
> 12/1/2022, 7:35:37.702 AM	Microsoft.ApiManagement/GatewayLogs	3c067a58-
> 12/1/2022, 7:35:57.510 AM	Microsoft.ApiManagement/GatewayLogs	974f270b-
∨ 12/1/2022, 7:34:51.086 AM	Microsoft.ApiManagement/GatewayLo...	e66e3fc5-
TenantId	fe27a755-2de8-45ae-ada3-27a35eac0c37	
TimeGenerated [UTC]	2022-12-01T15:34:51.0863278Z	
OperationName	Microsoft.ApiManagement/GatewayLogs	
CorrelationId	e66e3fc5-e127-45f7-b221-dbd67b9cfd14	
Region	East US	
IsRequestSuccess	false	
Category	Gateway logs	

Figura 2.2: Ejemplo de una bitácora de servicios

2.2 TEORÍA DE GRAFOS

Un grafo $G = (V, E)$ es una estructura matemática tipo red que consiste en un conjunto V de vértices y un conjunto E de aristas, donde $E \subseteq V \times V$. En este trabajo, utilizaremos los términos *nodo* y *conexión* para referirnos, respectivamente, a estos elementos. Ahora bien, un grafo se considera *dirigido* cuando para un par $(u, v) \in E$ no siempre existe su inverso (v, u) en E . Un grafo se considera *ponderado* cuando, para cada par $(u, v) \in E$, existe una etiqueta numérica $w(u, v)$ que representa el peso de la conexión; en este caso, además, se supone que $w(u, v)$ puede ser diferente de 1, que es el valor predeterminado en un grafo no ponderado. Si el grafo es ponderado y dirigido, $w(u, v)$ puede ser diferente a $w(v, u)$.

Cuando existe una conexión (u, v) o (v, u) donde $u \in E$ y $v \in E$, se considera que los nodos u y v son *adyacentes*, y también que son *vecinos*. Por lo tanto, para un nodo $v \in V$, su *vecindad* cuando $u \neq v$ está dada por $\Gamma(v) = \{u : (u, v) \in E\}$. Adicionalmente, en un grafo dirigido, se puede considerar que $\Gamma(v) = \Gamma_e(v) \cup \Gamma_s(v)$, donde $\Gamma_e(v) = \{u : (u, v) \in E\}$ es el *vecindario de entrada* y $\Gamma_s(v) = \{u : (v, u) \in E\}$ es el *vecindario de salida*.

Un nodo v también puede tener vecindarios de diferentes *órdenes*, o *saltos*, de tal manera que $\Gamma(v)$ es el vecindario de primer orden de v , y en órdenes superiores se incluye el vecindario de orden precedente:

$$\Gamma(v)^{i+1} = \Gamma(v)^i \cup \{u' : u' \in \Gamma(u), u \in \Gamma(v)^i\}, \quad (2.1)$$

donde u' representa un vecino de $i + 1$ saltos. Por ejemplo, para el vecindario de segundo orden (o un salto): $\Gamma(v)^2 = \Gamma(v) \cup \{u' : u' \in \Gamma(u), u \in \Gamma(v)\}$. De esta manera, si $\Gamma(v) = \{u, w, x\}$ y $\Gamma(u) = \{a, b, c\}$, entonces $\Gamma(v)^2 = \{u, w, x, a, b, c\}$. Asimismo, los vecindarios de diferentes órdenes también pueden descomponerse en vecindario de entrada y vecindario de salida. También se puede decir que, para un nodo v , él mismo sería su único vecino de 0-saltos (*0-hop neighbor*), sus vecinos “directos” serían vecinos de 1-salto (*1-hop neighbors*), sus vecinos de segundo orden serían vecinos de 2-saltos y, en general, sus vecinos de k orden serían vecinos de k -saltos.

2.2.1 REDES COMPLEJAS

Las *redes complejas* son grafos cuyas propiedades divergen de aquellas en un *grafo simple* (es decir, un grafo no dirigido y no ponderado). Estas redes representan tanto sistemas existentes en la naturaleza como sistemas creados por el ser humano. Algunos ejemplos incluyen a las redes de depredadores, las redes de transporte (metro, carreteras, vuelos), las redes de colaboración (por ejemplo, entre investigadores), las redes sociales en línea (por ejemplo, *Facebook*) y las redes de dispositivos móviles—entre muchas otras.

Algunas de las propiedades que distinguen a las redes complejas de los grafos simples incluyen el *mundo pequeño* y el *enlazado preferencial*. En una red de mundo pequeño, la mayoría de los nodos siguen caminos cortos para llegar a otros nodos (Newman, 2018); mientras tanto, en una red con enlazado preferencial, los nodos con muchas conexiones tienden a generar una mayor cantidad de estas a lo largo

del tiempo. Para cuantificar estas propiedades en los grafos, comúnmente se ha hecho uso de diferentes métricas, como centralidad (importancia), geodésica (camino más corto) y densidad. También se han desarrollado diferentes *modelos de grafos aleatorios* para simular redes complejas donde sobresalga alguna de las propiedades antes mencionadas. Por ejemplo, el modelo Watts-Strogatz representa redes con la propiedad de mundo pequeño (Newman, 2002).

El modelo Erdős-Rényi es uno de los primeros y más conocidos en la literatura. Este modelo recibe los parámetros n y p , donde n es la cantidad de nodos que tendrá el grafo aleatorio y p es la probabilidad de que se genere una conexión entre un par de nodos (se considera distribución uniforme, la probabilidad es la misma para todos los nodos). Entre mayor es el valor de p , más denso es el grafo generado —es decir, se tiene una mayor cantidad de conexiones. De esta manera, cuando $p = 0$, no existen conexiones en el grafo generado, y cuando $p = 1$ el grafo generado es un grafo completo (es decir, un grafo donde cada par de nodos es adyacente).

Otra variante del modelo Erdős-Rényi (que es la utilizada en este trabajo), consiste en emplear los parámetros n y m , donde m es la cantidad de conexiones que tendrá el grafo. En esta variante, por consiguiente, se tiene una cantidad fija de conexiones y solamente se determina de manera aleatoria cuáles serán los pares de nodos que unirán dichas conexiones.

Un modelo de grafo aleatorio que nos concierne también para el presente trabajo es el *modelo de configuración* (Newman, 2008), en el cual no se supone que la distribución de las conexiones sea paramétrica. En este modelo, se genera un grafo \mathcal{M} con la misma secuencia de grado de G , pero con los vecinos escogidos de manera aleatoria (con distribución uniforme) para cada nodo. La *secuencia de grado* de un grafo se refiere a una tupla ordenada que contiene todos los grados que existen en el grafo, donde por *grado* se entiende el número de conexiones de un nodo.

2.3 MINERÍA DE PROCESOS

En las organizaciones, los procesos de negocio no están exentos de presentar cuellos de botella y otras áreas de mejora. Para descubrir dichas áreas de mejora, y comprender con ello el rendimiento real de los sistemas, es posible extraer datos de las bitácoras que estos generan. La *minería de procesos* hace uso de estas bitácoras, ya que es una disciplina que estudia y propone técnicas para descubrir, validar y mejorar los flujos de trabajo en las organizaciones. Al combinar la minería de datos con el análisis de los procesos, la minería de procesos aprovecha un enfoque basado en datos para la optimización de estos procesos, lo que permite a los gerentes mantener la objetividad en su toma de decisiones sobre la asignación de recursos (Co., 2022).

La minería de procesos utiliza los registros o bitácoras de los sistemas de TI para modelar el día a día en una organización—descubriendo, así, divergencias con la planeación “en papel”. Por lo tanto, el rol de las bitácoras—extraídas, por ejemplo, de las herramientas de planificación de elementos empresariales (ERP) o de gestión de las relaciones de clientes (CRM)—es preponderante, pues ellas facilitan un seguimiento de auditoría de procesos. De esta manera, a partir de los datos en la bitácora, se examina el proceso de extremo a extremo y se describen los detalles de este, así como cualquier variación con respecto al diseño original (ver Figura 2.3).

Los algoritmos de minería de procesos también pueden proporcionar información sobre las causas fundamentales de las desviaciones del diseño planeado, y sirven para generar visualizaciones que permiten a la gerencia ver si sus procesos funcionan según lo previsto. De no ser así, ya cuentan con la información para justificar y asignar los recursos necesarios para su optimización. También pueden descubrir oportunidades para incorporar la automatización de procesos robóticos en los procesos, acelerando cualquier iniciativa de automatización para una empresa. De hecho, en 2011, el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) publicó el *Manifiesto de Minería de Procesos* (IEEE, 2022) en un esfuerzo por avanzar en la adopción

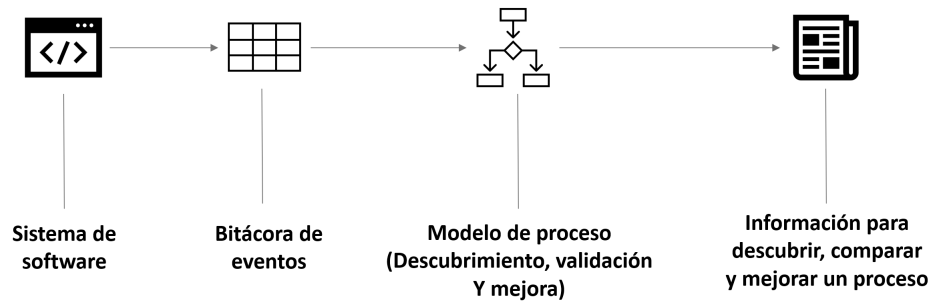


Figura 2.3: Flujo de la minería de procesos

de la minería de procesos para rediseñar las operaciones comerciales.

En la Figura 2.4, se muestra que dentro del flujo de la minería de procesos, se comienza con la operación de los sistemas de software, que generan bitácoras de esta operación. Las bitácoras se utilizan para generar el modelo de proceso que realmente ocurre en la compañía, y esta información se utiliza para comparar contra la planeación y así mejorar los procesos actuales. Adicionalmente, se puede apreciar los tres tipos de tareas que están relacionados con el uso de bitácoras en la minería de procesos:

1. *Descubrimiento de proceso.* Se basa en tomar una bitácora de eventos y generar el modelo de proceso correspondiente, sin tomar ninguna información adicional. De esta manera, se infiere un proceso que podría generar la bitácora que se está observando; típicamente, se producen redes de Petri para representar los procesos inferidos. Esta tarea es la más frecuente, pues las organizaciones pueden obtener ventaja de los registros obtenidos de sus sistemas para hacer descubrimientos que aporten valor en términos de mejora (dos Santos Garcia *et al.*, 2019).
2. *Validación de conformidad.* Compara un proceso existente contra un proceso inferido. Esta validación puede ser usada para verificar si la información almacenada es en realidad como se representa en el modelo de proceso y viceversa (Rajesh *et al.*, 2021).

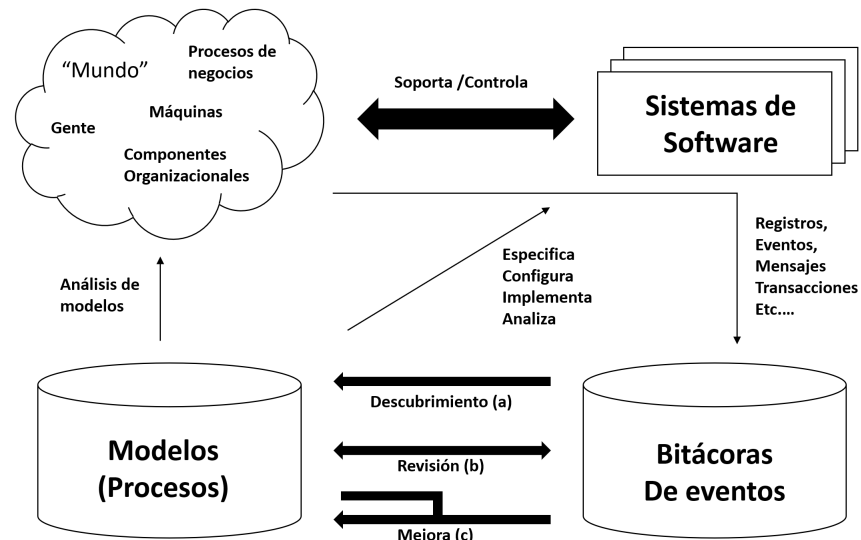


Figura 2.4: Representación del flujo de datos y procesos

3. *Mejoramiento*. Utiliza la información almacenada y revisada por la validación de conformidad para encontrar áreas de mejora. El objetivo de esta operación es cambiar o extender el proceso para eliminar cuellos de botella o mejorar los niveles de servicio (Zerbino *et al.*, 2021).

Vale la pena mencionar que, dentro de la minería de procesos, se han desarrollado diferentes algoritmos dedicados, principalmente, a la tarea de descubrimiento. El primero de estos algoritmos corresponde al *minero alfa* o " α -miner" (van der Aalst *et al.*, 2004), que utiliza reglas y condiciones para producir un modelo de proceso. Otro algoritmo bien conocido para hacer descubrimiento de procesos es el *minero heurístico* (Weijters *et al.*, 2006), el cual supone que las bitácoras pueden estar incompletas o contener ruido, y para ello se basa más bien en frecuencias y heurísticas para generar el modelo de proceso. Este algoritmo se describe con mayor profundidad en el Capítulo 3.

2.3.1 DEFINICIONES Y NOTACIÓN

Los siguientes conceptos son la base para el desarrollo de la presente investigación. Estos conceptos están sustentados en el marco teórico de minería de procesos. Se definen primero de manera intuitiva, y después se provee la notación correspondiente.

Un *proceso* representa el flujo ideal que describe un objetivo de negocio. Dentro de este contexto, una *actividad* consiste en una tarea posible y un *evento* es una actividad individual que se realiza en la ejecución de un proceso. Un evento posee *propiedades*, tales como identificador, nombre de evento, firma de tiempo y actor que corresponde a la actividad ejecutada, entre otros (ver la Figura 2.5). Una *traza* (también conocida como *caso*) representa la ejecución de un proceso, y puede definirse más concretamente como una secuencia de eventos. A su vez, una *bitácora de eventos* (*log of events*) puede verse como una secuencia de trazas. Sin embargo, es posible que el conjunto de propiedades de los eventos no incluya un *identificador de caso* (o *identificador de traza*) que los asocie con una traza en particular. A este tipo de bitácora, se le conoce como *bitácora no etiquetada* o *bitácora sin casos* (*caseless event log*). A continuación, se describen de manera formal estos conceptos.

Consideremos un universo A de actividades y un universo \mathcal{E} de eventos. Un evento $e \in \mathcal{E}$ puede verse como la ocurrencia de una actividad, y está representado por una terna de propiedades (a, t, χ) , donde:

- $a \in A$ es la actividad a la que corresponde e ,
- t es la firma de tiempo registrada,
- χ es el identificador de caso al cual pertenece e .

Para cada propiedad de e , la función $\#_p(e) = p$ devuelve la propiedad p , de tal manera que

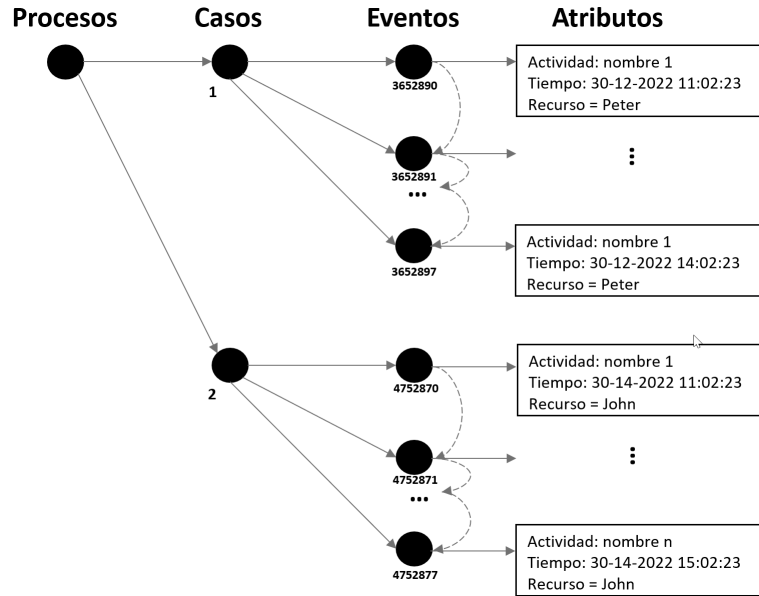


Figura 2.5: Estructura de un registro de eventos.

- $\#_1(e) = a$,
- $\#_2(e) = t$ y
- $\#_3(e) = \chi$.

Cuando la propiedad tiene un valor nulo para el evento, entonces $\#_p(e) = \perp$.

Los eventos que tienen el mismo identificador de caso pertenecen a la misma *traza*, donde una traza σ es una secuencia finita de eventos, de tal manera que $\sigma = \langle e : e \in \mathcal{E} \rangle$ donde $\sigma \in e \times e \times e \dots \times e_n$. Cada traza consiste de n posiciones, tal que $n = |\sigma|$ y $\sigma(i) = e_i$ representa la *isesima* componente de σ y está en la posición i de la traza σ .

De esta forma, una bitácora de eventos \mathcal{L} viene a ser una secuencia finita de trazas, tal que $\mathcal{L} = \langle \sigma : \sigma \in \mathcal{E}^* \rangle$ y $\mathcal{L} \in \sigma \times \sigma \times \sigma \dots \times \sigma_n$, donde \mathcal{E}^* es el universo de posibles trazas sobre \mathcal{E} .

En \mathcal{L} , las relaciones de precedencia entre un par de eventos s y t pueden cuantificarse a través de dos medidas: la *medida de secuencia* y la *medida de dependencia*.

Estas dos medidas se describen, respectivamente, en las Definiciones 1 y 2.

Definición 1 *Medida de secuencia entre eventos.*

Sea \mathcal{L} una bitácora de eventos sobre el universo \mathcal{E}^* de secuencias finitas de trazas, y sea además que $s, t \in \mathcal{E}$. La medida de secuencia $|s >_{\mathcal{L}} t|$ es el número de veces que s es inmediatamente seguido por t en \mathcal{L} . Podemos expresar esta medida de la siguiente forma:

$$|s >_{\mathcal{L}} t| = \sum_{\sigma \in \mathcal{L}} f_{\mathcal{L}}(\sigma) \cdot |\{\pi(s, t) : s = \sigma(i) \wedge t = \sigma(i+1)\}|, \quad (2.2)$$

donde establecemos que:

- $i > 0$ y $i < |\sigma| - 1$
- $f_{\mathcal{L}}(\sigma)$ representa la frecuencia de la traza σ en \mathcal{L} y
- $\pi(s, t)$ representa una ocurrencia de precedencia inmediata entre los eventos s y t .

Definición 2 *Medida de dependencia entre eventos.*

Definimos $|s \Rightarrow_{\mathcal{L}} t|$ como el valor de dependencia entre s y t , y lo calculamos de la siguiente manera:

$$|s \Rightarrow_{\mathcal{L}} t| = \begin{cases} \frac{|s >_{\mathcal{L}} s|}{|s >_{\mathcal{L}} s| + 1} & \text{si } s = t \\ \frac{|s >_{\mathcal{L}} t| - |t >_{\mathcal{L}} s|}{|s >_{\mathcal{L}} t| + |t >_{\mathcal{L}} s| + 1} & \text{si } s \neq t \end{cases} \quad (2.3)$$

Cabe destacar que $|s \Rightarrow_{\mathcal{L}} t| \in [-1, 1]$. Si el valor es cercano a 1, entonces existe una dependencia fuerte y positiva entre s y t , por lo cual s comúnmente causaría t . Este valor podría obtenerse cuando s precede a t , pero t generalmente no precede a s . Por otra parte, un valor cercano a -1 indica una dependencia fuerte y negativa entre s y t , y t comúnmente causaría s . Un valor cercano a 0 indicaría que existe dependencia muy débil entre s y t .

Basándonos en las Definiciones 1 y 2, podemos definir el *grafo de secuencia* y el *grafo de dependencia* (Definición 3 y Ecuación 2.5).

Definición 3 *Grafo de secuencia.* Un grafo de secuencia es un grafo dirigido y ponderado donde los nodos representan eventos y los pesos en las conexiones representan la medida de secuencia entre un par de eventos. Formalmente: $\mathcal{S} = (V, E)$, $V = \{e : e \in \mathcal{E}\}$, $E = \{(s, t) : s, t \in \mathcal{E} \wedge w(s, t) \geq \tau_S\}$, y

$$w(s, t) = |s \succ_{\mathcal{L}} t|, \quad (2.4)$$

donde $w(s, t)$ es el peso de la conexión entre s y t , y τ_S representa un umbral.

De manera similar, el grafo de dependencia corresponde a un grafo dirigido y ponderado donde los pesos de las conexiones están dados por la medida de dependencia, de tal suerte que

$$w(s, t) = |s \Rightarrow_{\mathcal{L}} t|. \quad (2.5)$$

2.4 RESUMEN

En el presente capítulo, se describieron los conceptos más relevantes para la correcta comprensión de nuestra solución propuesta ante el problema de descubrimiento de relaciones entre microservicios. Primeramente, vimos que los microservicios son componentes atómicos con una funcionalidad específica. En segundo lugar, vimos terminología perteneciente a la teoría de grafos; dentro de dicha terminología, se repasaron conceptos como *nodo*, *conexión*, *grafo ponderado* y *vecindad de k orden*; de igual manera, se explicó brevemente el *modelo de configuración* para generar redes complejas de manera aleatoria. Por último, se repasaron conceptos pertenecientes a la minería de procesos, la cual tiene como objetivo el descubrimiento, evaluación y mejora de los flujos de trabajo en las organizaciones. Algunos de los conceptos repasados incluyen *actividad*, *evento*, *traza*, *bitácora de eventos*, *medida de secuencia* y *grafo de dependencia*.

CAPÍTULO 3

ESTADO DEL ARTE

Dentro del estado del arte, se puede identificar varias ramas de contribución, ya que el método propuesto se basa en la adaptación de herramientas y disciplinas diversas. Estas incluyen la minería de procesos, el descubrimiento de relaciones con grafos, y las arquitecturas basadas en microservicios. Con esto en mente, el presente capítulo se divide precisamente en esas contribuciones para cada una de las ramas mencionadas. El alcance del presente capítulo se enmarca en aquellos trabajos que son relevantes en diferentes grados, ya sea por el método de resolución o por la problemática que abordan.

3.1 MINERÍA DE PROCESOS

Respecto a la minería de procesos, nuestro problema está enfocado en la tarea de descubrimiento de procesos. A los algoritmos de descubrimiento se les suele llamar *mineros*, y dado que esta tarea está lejos de ser trivial por la complejidad que representa, se han desarrollado múltiples mineros que ofrecen solución para diferentes escenarios. Cada uno de estos tiene ventajas y desventajas, con mayor o menor tolerancia al ruido de los datos. En la Figura 3.1 se listan los mineros más relevantes y las investigaciones que mejor representan el problema que se aborda en el presente documento. Sin embargo, hay diferencias significativas dentro del tipo de

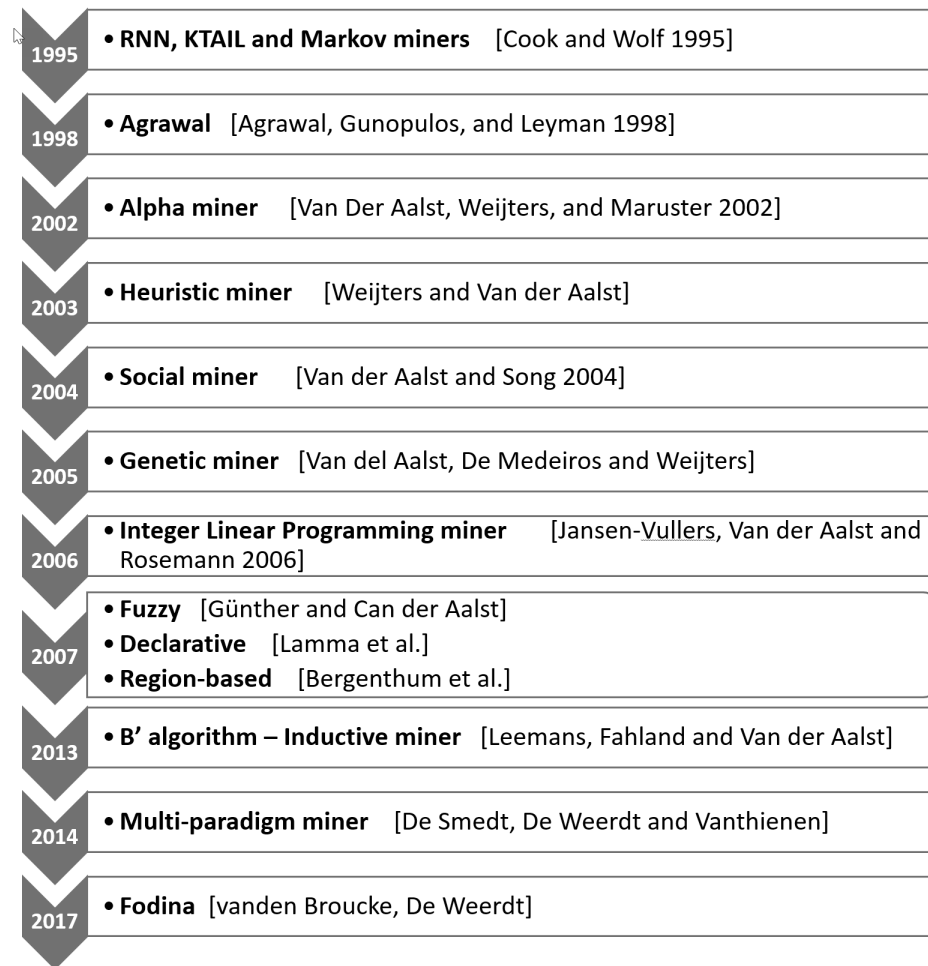


Figura 3.1: Principales algoritmos de minería

planteamiento del problema, los datos disponibles y la forma en que se aborda la solución.

ALGORITMOS MINEROS. El *minero alfa* (“Alpha Miner”) fue presentado por primera vez por van der Aalst *et al.* (2004), y tiene como objetivo el generar modelos de proceso mediante la reconstrucción de causalidad a partir de un conjunto de secuencias de eventos. Esto significa que el propósito de este minero es transformar el registro de eventos en una red de trabajo que se basa en las conexiones entre las actividades; esta red de trabajo se muestra como una red de Petri. El minero alfa fue el primer método de descubrimiento de procesos que se sugirió y ofrece una buena visión general del objetivo del descubrimiento de procesos y cómo se realizan varias

actividades dentro del proceso. También fue el fundamento para el desarrollo de otras técnicas de minería de procesos, como el minero heurístico y el minero genético.

El *minero heurístico* (Weijters *et al.*, 2006), a diferencia del minero alfa, considera que la bitácora de eventos puede ser imprecisa y contener ruido. Por lo mismo, extrae la perspectiva del flujo de trabajo de un modelo de proceso considerando el orden de los eventos dentro de un caso. En ese sentido, trabaja con las propiedades de identificador de traza, firma de tiempo y evento (actividad) durante la extracción. La firma de tiempo de un evento se usa para calcular el orden de estos eventos dentro de la traza. El punto inicial del minero heurístico es la creación de un grafo de dependencia. Se utiliza una métrica basada en la frecuencia para indicar la certidumbre de la relación de dependencia entre dos eventos a y b .

MINEROS CON INFERENCIA DE IDENTIFICADORES DE TRAZA. Los primeros mineros suponen que todos los eventos cuentan con un identificador de traza. Sin embargo, esto no siempre es así. Es por esta razón, que también se han construido mineros que infieren el identificador de traza en bitácoras no etiquetadas. Por ejemplo, Bayomie *et al.* (2016) enfrentan este reto mediante un enfoque basado en heurísticas para realizar la inferencia de estos identificadores. Este enfoque requiere, no obstante, el modelo planeado del proceso como entrada adicional. La heurística utilizada por estos autores se basa en los tiempos de ejecución de los eventos. La salida de este algoritmo es un conjunto de eventos con el identificador inferido y una puntuación que indica el nivel de confianza de que este identificador sea el correcto.

Recientemente, Bayomie *et al.* (2019) abordan la problemática de las bitácoras no etiquetadas mediante un enfoque probabilístico. Este enfoque plantea el problema como un problema de optimización multinivel, proponiendo un algoritmo denominado EC-SA (*Simulated Annealing Event Correlation*), que se basa en el recocido simulado. Este método de mejora se basa en dos metas encadenadas: primero, trata de reducir la falta de correspondencia entre el registro y un modelo de proceso dado; segundo, trata de reducir la variación en la duración de la actividad entre rastros.

Esta última meta se apoya en la premisa de que las mismas actividades suelen tener una duración parecida en todos los rastros. El algoritmo toma como entrada un conjunto de eventos sin identificadores de traza y un modelo de proceso (normativo o descriptivo) que captura el conocimiento del proceso comercial subyacente. Produce como salida los identificadores de traza.

REVISIONES Y APLICACIONES. El campo de la ciberseguridad ha mostrado interés en el descubrimiento de modelos de minería de procesos, una técnica que permite revelar los patrones de las estrategias de ciberataque a partir de un registro de alertas de intrusión. Esto, a su vez, permite descubrir anomalías en los procesos de ciberseguridad y permite lidiar con formas de ciberataque más nuevas y más sofisticadas, que son frecuentes en redes empresariales. Por ejemplo, el trabajo desarrollado por Jacob *et al.* (2019) busca responder la pregunta: *¿De qué manera puede la minería de procesos de negocios mejorar la detección de ataques de seguridad cibernética en un dominio basado en microservicios?*

Otra aplicación interesante es la *descomposición funcional* durante la migración de sistemas, pues esta consiste en pasar de un solo bloque de código (arquitectura monolítica) a múltiples bloques (arquitectura basada en microservicios). Dicha tarea es compleja, ya que generalmente se realiza de manera manual según la experiencia de los arquitectos de software. Taibi y Systä (2019) proponen un modelo de trabajo que consta de seis pasos para disminuir el problema de la subjetividad de esta tarea. Las opciones de descomposición se identifican en función de los rastros de ejecución independientes del sistema mediante la aplicación de una herramienta de minería de procesos a los rastros recopilados en tiempo de ejecución desde una bitácora. La empresa pudo detectar fallos en su software que el arquitecto no había encontrado manualmente y explorar alternativas de descomposición más apropiadas que el arquitecto no había tenido en cuenta, gracias a los experimentos realizados con el marco propuesto.

También se ha estudiado el uso de redes de Petri para microservicios. Por

ejemplo, Sakai *et al.* (2021) estiman la dependencia entre servicios y la expresan como un modelo de red de Petri para manejar los procesos no deterministas de servicios. Se evalúa la aplicabilidad del modelado a servicios con este tipo de comportamiento y se confirma que se pueden modelar varios procesos con el método propuesto.

En estudios recientes, donde se describe el estado actual respecto la minería de procesos, dos Santos Garcia *et al.* (2019) describen las principales editoriales por país, publicaciones periódicas y congresos. También se obtienen los estudios de aplicación basados en información y se clasifican por dominios de exploración o también de segmentos de la industria que utilizan esta técnica. Este trabajo nos permite tener una visión más clara de la adopción de esta área.

Un artículo revisa la investigación en gestión sobre minería de procesos y administración de negocios con el fin de evaluar el estado del arte y sugerir direcciones para futuros estudios (Zerbino *et al.*, 2021). Para ello, se propone un marco de siete dimensiones que orienta y estructura la revisión. Se escogen y examinan 145 artículos y se detectan once brechas de investigación agrupadas en cuatro categorías.

3.2 MEJORAMIENTO DE ARQUITECTURAS BASADAS EN MICROSERVICIOS

Debido a que las arquitecturas basadas en microservicios aportan distintas ventajas a las organizaciones (e.g. flexibilidad), se trata continuamente de mejorarlas y optimizarlas. Mientras que algunos trabajos buscan generar bloques de microservicios, otros buscan estrategias de migración hacia arquitecturas orientadas a servicios. También existe una serie de trabajos dedicados a mejorar la búsqueda de servicios aptos para tareas particulares.

GENERACIÓN DE FLUJOS EN ARQUITECTURAS DE MICROSERVICIOS. En la *generación de flujos* a base de microservicios, se busca desarrollar componentes (“blo-

ques”) de software que sean más granulares y sirvan para diferentes aplicaciones—sin necesidad de armar la arquitectura partiendo solamente de servicios individuales. Por ejemplo, el trabajo de Oberhauser y Stigler (2018), considerando los desafíos de automatización, rapidez y flexibilidad, propone un modelado ágil de procesos comerciales con *micro flujos*, los cuales se describen como un enfoque declarativo ligero automático para la orquestación centrada en el flujo de trabajo de microservicios; para ello, utilizan anotaciones semánticas mediante clientes basados en agentes, métodos basados en grafos y vocabularios semánticos ligeros (JSON-LD y Hydra). Por otra parte, Munonye (2021) hace uso de un modelo basado en los principios de la minería de datos llamado *stream analytics feedback and optimization* (SAFAO). Para ello, realiza un análisis de flujo en una aplicación en línea con datos en vivo generados de manera continua.

DESCOMPOSICIÓN MEDIANTE MICROSERVICIOS. Además de brindar una mayor flexibilidad, las arquitecturas basadas en microservicios también ayudan a generar software más entendible y mantenible. Por esta razón, se ha llevado a cabo—desde hace ya tiempo—la *descomposición* de arquitecturas monolíticas en arquitecturas orientadas a servicios. Esta descomposición, como se mencionó previamente, tradicionalmente se ha hecho de forma manual, ya que su automatización conlleva diferentes retos. Por ejemplo, los métodos existentes para la descomposición automática de servicios web en muchos casos se basan en la comparación de los parámetros de entrada y salida, por lo que están limitados a árboles de composición y tienen problemas para armar redes debido a que ignoran la relación entre ellos. Para abordar este problema, Ying (2010) propone un modelo que reduce un grafo dirigido a una representación con nodos compartidos. Por tanto, en este método, se modela la composición de servicios como un grafo dirigido, y el grafo de composición de servicios máximo se construye primero, luego todos los grafos de composición mínima se abstraerán del máximo por los nodos compartidos en él. Este método puede manejar el plan de composición lineal-árbol-red porque descubre no sólo todos los servicios que participan en la composición, sino también todas las relaciones entre servicios.

De igual manera, existen investigaciones que se basan en evaluar métodos utilizados en la industria para descomponer sistemas monolíticos a propuesta de composición de servicio, siendo una de estas investigaciones la de Kirby *et al.* (2021), que en sus resultados encontró que una herramienta de validación de escenarios sigue siendo necesaria. Los principales puntos en que se basó este análisis fueron: (1) la aplicabilidad y la utilidad de diferentes tipos de relaciones durante el proceso de extracción de microservicios y (2) expectativas que tienen los profesionales sobre las herramientas que utilizan dichas relaciones.

DESCUBRIMIENTO DE MICROSERVICIOS. El descubrimiento de servicios (también conocido como “análisis semántico”) trata de localizar microservicios que sean compatibles uno con otro. Esto permite realizar diferentes tareas, como composición y generación de flujos (vistas anteriormente). Sin embargo, vale la pena aclarar que el descubrimiento de servicios es diferente al descubrimiento de *relaciones* entre microservicios (que es el problema tratado en el presente trabajo). Aún así, ambas tareas representan un reto para el mejoramiento de las arquitecturas basadas en servicios.

Dentro de la línea de descubrimiento de servicios, hay una investigación respecto a la exploración de modelos de la semántica para servicios web a través de un experimento dentro de los sujetos y se amplía la metodología con investigaciones actuales sobre procedencia, web semántica y estándares de servicios web, desarrollando y evaluando empíricamente un enfoque integrado para la descripción y el descubrimiento de servicios web (Narock *et al.*, 2014). Además, se discuten las consecuencias para algoritmos más sofisticados de hallazgo de servicios web e interfaces de usuario. Adicionalmente, hay trabajos que se basan en la resolución del problema de encontrar las relaciones dinámicas en el descubrimiento de eventos. Para detectar esos eventos, los principales retos son encontrar y confirmar las relaciones dinámicas entre las entidades, y agrupar las relaciones dinámicas binarias en eventos que involucran a un conjunto de entidades relacionadas en un intervalo de tiempo específico (Das Sarma *et al.*, 2011).

Dentro de los trabajos sobre el uso semántico en servicios web, el contexto es útil para encontrar servicios que satisfagan los requisitos de los usuarios con mayor precisión. El contexto, que se clasifica *servicio* y *usuario*, se utiliza por Liu *et al.* (2018) dentro de un algoritmo de descubrimiento de microservicios agrupados en ese contexto. En primer lugar, los microservicios se agrupan de acuerdo con la similitud del contexto del servicio y se inicializa un conjunto de servicios candidatos al hacer coincidir la solicitud con los grupos de servicios. Luego, de acuerdo con la similitud del contexto del usuario, los usuarios que usaron y están usando los servicios candidatos se agrupan y el conjunto de servicios candidatos se refina haciendo coincidir la información del solicitante con el contexto de los usuarios en estos grupos. Finalmente, la calidad de servicio (QoS) y la preferencia del solicitante se utilizan para filtrar los servicios candidatos y los resultados se invierten, se indexan y se devuelven al solicitante.

Respecto al análisis semántico de las propiedades de los servicios web, Omer y Schill (2009) proponen un método para la composición automática que se basa en la extracción de dependencias entre servicios. Esta extracción se basa en las similitudes semánticas de los parámetros de entrada y salida de los servicios, y estas dependencias son representadas en grafo dirigido. El método detecta las dependencias cíclicas que puedan existir y sugiere una manera de resolverlas. Usando este método modificado de ordenación topológica, se genera el plan de ejecución que indica el orden en que se ejecutan los servicios candidatos. El trabajo de Ben Lamine *et al.* (2017), asimismo, propone un método de descubrimiento colaborativo semántico de servicios RESTful bajo el principio de HATEOAS. El método tiene el siguiente alcance: (1) ligas semánticas entre servicios, (2) agrupamiento de perfiles de usuario en base a la media de similaridad, y (3) ontología de perfil de usuario y los correspondientes servicios.

En el área de clasificación para descubrimiento de servicios web, Li *et al.* (2021) presentan redes por convolución para generar un modelo de clasificación llamado *Red convolucional – Grafo de atención residual*. Se usa un mecanismo de atención con la

red convolucional y aprendizaje residual para aumentar la profundidad del modelo y extraer más características.

3.3 DESCUBRIMIENTO DE RELACIONES

El descubrimiento de relaciones no es exclusivo de las arquitecturas orientadas a servicios. Por consiguiente, es posible encontrar en la literatura trabajos que identifican relaciones entre otros tipos de elementos, tales como texto o peticiones de HTTP. También existen trabajos que se enfocan en tareas similares, como la predicción de enlaces en grafos.

MINERÍA DE REGLAS DE ASOCIACIÓN. En cuanto al descubrimiento de relaciones (sin ahondar necesariamente en microservicios), se han propuesto trabajos de muy diversa índole. Por ejemplo, se han propuesto algoritmos de minería de reglas de asociación para extraer conjuntos de elementos frecuentes (*frequent itemsets*) en un cubo de datos (Singh *et al.*, 2015). El algoritmo propuesto descubre estos conjuntos mediante la función de agregación y el uso de un grafo dirigido, además de ahorrar consumo de memoria en la generación de candidatos. Asimismo, utiliza una función de agregación para la reducción de dimensiones y el grafo dirigido para las generaciones de conjuntos de elementos candidatos.

VINCULACIÓN TEXTUAL. La vinculación textual (también conocida como *inferencia de lenguaje* o *textual entailment* en inglés) es una tarea propia del procesamiento de lenguaje natural que descubre la relación entre fragmentos de texto. Los sistemas de vinculación textual generalmente están diseñados para trabajar con palabras sueltas, relaciones u oraciones completas. La detección de vínculos entre fragmentos de grafos de dependencia de cualquier tipo, que relaja estas restricciones, conduce a un descubrimiento de vínculos mucho más amplio. En ese sentido, en el trabajo de Rei y Briscoe (2011) se describe un marco no supervisado que utiliza la similitud

intrínseca, la similitud extrínseca de varios niveles, la detección de la negación y el lenguaje protegido para asignar una puntuación de confianza a las relaciones de vinculación entre dos fragmentos.

RELACIONES TEMPORALES. En el estudio de Das Sarma *et al.* (2011), se aborda el problema de la relación dinámica y el descubrimiento de eventos. Su objetivo es identificar las relaciones definidas temporalmente que no están predefinidas por un esquema existente, e identifica los eventos subyacentes limitados en el tiempo que conducen a estas relaciones. El proceso para hallar estos eventos consiste en encontrar y confirmar vínculos dinámicos entre entidades y unir vínculos dinámicos binarios en eventos que abarcan un conjunto de entidades que están relacionadas en un intervalo de tiempo específico.

PREDICCIÓN DE RELACIONES. Un problema relacionado con el descubrimiento de relaciones es la *predicción de relaciones* (en inglés, *link prediction*), la cual consiste en detectar de manera anticipada los enlaces que aparecerán en un grafo con el paso del tiempo. Berton *et al.* (2015) proponen, para esta tarea, los siguientes pasos: (i) aumentar la exploración de algoritmos basados en grafos y (ii) construir grafos a partir de datos planos mediante nuevas técnicas. Su propuesta se enfoca en la construcción de grafos que tengan medidas de predicción de enlaces como base. A partir de una estructura gráfica básica, se aplica esta medida para agregar nuevas conexiones en el grafo. Los autores utilizan criterios de predicción de enlaces que se basan en la semejanza estructural del grafo que incrementa la cohesión del grafo.

DESCUBRIMIENTO EN LA NUBE. Una aplicación con respecto al descubrimiento de relaciones, es propuesta en la investigación de Fang *et al.* (2014) que describe la creación de un grafo de dependencia para describir la relación entre peticiones HTTP. Este modelo se basa en una heurística paralela que distingue los clics de los usuarios con la asistencia de cómputo en la nube.

3.4 GRAFOS APLICADOS AL DESARROLLO DE SOFTWARE

Los grafos han sido una estructura comúnmente utilizada en el desarrollo de software. Un ejemplo de lo anterior es el manejo de ecosistemas de versiones, donde los grafos se han empleado para manejar cambios y minimizar el impacto de estos cambios. En el área de microservicios, los grafos se han utilizado—por ejemplo—para generar aplicaciones relacionadas con el Internet de las Cosas. Los grafos de dependencia, por otro lado, han servido para análisis de fallas y anomalías.

MANEJO DE LIBRERÍAS, VERSIONES Y API. Los sistemas de código abierto basados en librerías y repositorios, como **Maven** o **NPM**, declaran dependencias a librerías externas y a herramientas automáticas que las hacen disponibles a entornos de trabajo. Sin embargo, esto acarrea brechas de seguridad como **EquiFax** o la remoción del paquete **leftpad** en versiones posteriores. Existen investigaciones que proponen una red de dependencia a nivel fino que va más allá de los paquetes (Hejderup *et al.*, 2018), dando como resultado un grafo de versiones del ecosistema a diferentes niveles.

La documentación de las API (aplicación de interfaz de usuario) proporciona información importante sobre la funcionalidad y el uso de estas. El trabajo de **Stackoverflow** (2022) se centra en los mensajes de advertencia sobre el uso API que los desarrolladores deben tener en cuenta para evitar el uso involuntario de una API en particular; con el soporte de las preguntas del sitio de **StackOverflow**, los autores demuestran que estas advertencias de uso de APIs suelen estar dispersas en varios documentos y se encuentran en largas descripciones textuales. Siguiendo esta misma línea, en la investigación de Li *et al.* (2018), se proponen técnicas de procesamiento de lenguaje natural (NLP) para extraer diez sub-categorías de oraciones de advertencia de API en la documentación y vincular estas oraciones a entidades en un

grafo de conocimiento. Este grafo de conocimiento de las advertencias de la API puede admitir la búsqueda de advertencias basada en recuperación de información (*information retrieval*) o centrada en la entidad.

GRAFOS PARA MICROSERVICIOS. Existen trabajos orientados al uso de grafos dirigidos en combinación con microservicios, como el caso de Ibrahim *et al.* (2019), donde se proponen grafos de ataque como parte de las infraestructuras basadas en microservicios. Con esto como fin, se analiza la red de microservicios para automáticamente generar este grafo de ataque que será incluido como parte del proceso de entrega continua.

Dada la complejidad de las arquitecturas basadas en servicios, es difícil entender el gobierno y el vasto ecosistema de una aplicación, ya que esta evoluciona de manera independiente y diferente. Como se vio con el trabajo anterior, existen también modelos para representar este ecosistema de servicios en un grafo de conocimiento con la intención para dar visibilidad (Wang *et al.*, 2019). Ahora bien, respecto a la composición óptima de servicios bajo las restricciones de calidad en el servicio (QoS), Chao y Meng-ting (2010) proponen un algoritmo BTWS para la selección de los servicios web, el cual a su vez se basa en un grafo de función ordenada. Con este grafo, se convierte el problema en un problema de teoría de grafos basado en la ruta óptima con múltiples restricciones.

Asimismo, la arquitectura basada en servicios puede tener una complejidad inherente que impide a los usuarios realizar un seguimiento y una gestión adecuados. Particularmente, es muy complicado identificar la causa raíz de una anomalía una vez que se detecta en la aplicación, y puede ser una tarea difícil y lenta, considerando el total de los servicios y conexiones disponibles que se tienen que revisar. La investigación por parte de Álvaro Brandón *et al.* (2020) presenta un marco de análisis de causa raíz, que se basa en representaciones basadas en grafos de estas arquitecturas. La comparación de grafos puede ayudar a detectar situaciones anómalas que se presenten en el sistema, utilizando una colección de grafos anómalos como

referencia para que el usuario pueda resolver dichos problemas.

Una forma de entender el Internet de las Cosas (IoT, por sus siglas en inglés) es como una arquitectura basada en servicios —en particular microservicios. Los microservicios procesan inherentemente datos que afectan la privacidad y la seguridad de sus usuarios. La seguridad del servicio IoT es un desafío clave. La mayor parte del estado del arte que proporciona seguridad del sistema IoT se basa en políticas. En la propuesta de Pahl *et al.* (2018) se muestra un control de acceso basado en grafos son ejecutados como módulo en nodos de IoT o en la red. El modelo propuesto esta basado en la interceptación y bloqueo de la comunicación entre servicios. Este proceso genera un modelo automático de las relaciones de comunicación válidas. El modelo se modifica de manera interactiva mediante una interfaz sencilla.

USO DE BITÁCORAS. Respecto al problema en el que las administraciones de servicios intentan monitorear el rendimiento y el estado de los servicios mediante bitácoras, hay trabajos que proponen extraer registros no estructurados como patrones de registro y visualizar las relaciones temporales de los servicios mediante grafos temporales. El método de análisis y minería de datos propuesto por Zuo *et al.* (2021) pretende brindar nuevos conocimientos en el área de administración del sistema y el proceso de análisis no trivial. En su trabajo, también Zhu *et al.* (2019) evalúan trece diferentes analizadores de bitácoras para informar los resultados de la evaluación comparativa en términos de precisión, solidez y eficiencia, que son de importancia práctica al implementar el análisis de registros automatizado en producción.

Para abordar la gran cantidad de seguimientos producidos en el tiempo de ejecución por microservicios, Guo *et al.* (2020) proponen desarrollar un enfoque de análisis de seguimiento de microservicios basado en grafos GMTA para comprender la arquitectura y diagnosticar varios problemas. Basado en una representación de grafos, GMTA realiza un procesamiento eficaz de las trazas que se generan dinámicamente. Simplifica las trazas en distintas rutas y las clasifica en flujos de negocio.

USO DE GRAFOS DE DEPENDENCIA. A pesar de los métodos de análisis semántico de software, se prefieren métodos que generalmente tienen en común problemas con complejidad en tiempo y resultados no precisos. (Meng *et al.*, 2015) proponen un algoritmo de programa controlado de flujo que se basa en grafos de dependencia de control y la abstracción de un árbol de sintaxis, este algoritmo mejora el proceso tradicional de construcción de un grafo de dependencia.

Dentro del área de análisis en tiempo real, hay propuestas para desarrollar un grafo en el vuelo basado en trazas de la operación generados por los microservicios (Guo *et al.*, 2020). En este trabajo, se abstraen las diferentes trazas dentro de diferentes caminos y grupos para agruparlos en flujos de negocios. El análisis de trazas de basa en la dependencia de campo del SpanID para determinar dependencias.

Respecto al uso de grafos en el área de pruebas, Wang y Zhang (2017) proponen la generación de un módulo de grafo de dependencia combinado con un módulo de interacción de la información para el procesamiento de las pruebas de integración. Lo anterior, desde una perspectiva orientada a aspectos basada en dos niveles: módulo y dependencia de datos. Por otra parte, en la investigación sobre el análisis de patrones sobre un elemento dado de una red (Ni *et al.*, 2020), se definió un marco de trabajo que tiene dos pasos, que se ejecutan de manera iterativa. Uno tiene el objetivo de expandir el conjunto de elementos de dependencia local que inicialmente consta únicamente del elemento dado; el otro se enfoca en actualizar la red de productos locales. Este marco de trabajo se aplica usando tres medidas distintas de dependencia y un algoritmo común de identificación de la comunidad local.

El uso de grafos de dependencia para analizar y probar microservicios es propuesto por Ma *et al.* (2018) dentro de un enfoque para ayudar al desarrollo de sistemas basados en arquitecturas de microservicios. Este grafo es denominado GMAT (*análisis y prueba de microservicios basados en grafos*). Una forma de ver o examinar cómo se relacionan los microservicios entre sí es mediante un grafo de dependencia del servicio (SDG), que GMAT puede crear de forma automática. Usando GMAT,

las anomalías podrían detectarse mediante el análisis de cadenas de invocación de servicios en riesgo en las primeras fases de desarrollo y rastrear los vínculos entre los servicios al desarrollar una nueva versión.

No es posible caracterizar las anomalías que se producen en diferentes servicios con un solo tipo de métrica, por lo que los algoritmos que se basan en una única métrica suelen fallar a la hora de encontrar la causa original de la anomalía. Con relación a esta problemática, Ma *et al.* (2020) proponen una herramienta llamada AutoMAP, que permite la generación dinámica de correlaciones de servicios y diagnósticos automatizados aprovechando múltiples tipos de métricas. AutoMAP se basa en el concepto de *grafo de comportamiento anómalo* para representar las relaciones entre servicios que involucran distintos tipos de métricas. Se definen dos operaciones binarias y una función de similitud en el grafo de comportamiento para facilitar a AutoMAP la selección de la métrica de diagnóstico más apropiada en cada situación específica. Siguiendo el grafo de comportamiento, se diseñó un algoritmo de investigación heurística mediante el uso de caminatas aleatorias hacia atrás y hacia adelante, con el objetivo de identificar la causa raíz de los servicios.

3.5 RESUMEN

A lo largo de este capítulo hemos revisado trabajos relevantes para la presente investigación. En las Tablas 3.1, 3.2 y 3.3 se hace una sinopsis de los enfoques y aplicaciones. Sin embargo, es necesario recalcar que la naturaleza de la misma implica la convergencia de muchos aspectos de cada una de las áreas mencionadas y por lo mismo, si bien hay trabajos que parecieran atacar el mismo problema, la base de donde se parte es distinta.

Los trabajos relacionados con la minería de procesos nos ofrecen una perspectiva de generación de proceso, y nosotros buscamos llegar solamente a la primera parte relevante de esa disciplina: revelar las relaciones entre eventos a través de al-

Tabla 3.1: Trabajos relacionados con minería de procesos

Trabajo	Tema	Propuesta
Bayomie <i>et al.</i> (2016)	Inferencia identificadores	Método heurístico
Bayomie <i>et al.</i> (2019)	Inferencia identificadores	Método probabilístico
dos Santos Garcia <i>et al.</i> (2019)	Revisiones y aplicaciones	Técnicas de minería de procesos
Jacob <i>et al.</i> (2019)	Revisiones y aplicaciones	Detección de Ataques
Sakai <i>et al.</i> (2021)	Revisiones y aplicaciones	Modelado de Procesos con redes de Petri
Taibi y Systä (2019)	Descomposición de Procesos	Reducción de Subjetividad
van der Aalst <i>et al.</i> (2004)	Algoritmos mineros	Modelado de procesos con reglas
Weijters <i>et al.</i> (2006)	Algoritmos mineros	Modelado de procesos con heurísticas
Zerbino <i>et al.</i> (2021)	Revisión Bibliográfica	Minería de Procesos y administración de Negocios

go mensurable. Por otro lado, la teoría de grafos nos da un margen muy amplio de herramientas para poder comprobar la hipótesis sobre la existencia de aristas y la valoración de rutas, pero son las investigaciones sobre microservicios y descubrimiento de relaciones las que finalmente nos ayudan a dar sentido a las herramientas provistas por las anteriores áreas.

Tabla 3.2: Trabajos relacionados con mejoramiento de arquitecturas basadas en microservicios

Trabajo	Tema	Propuesta
Ben Lamine <i>et al.</i> (2017)	Descubrimiento	Descubrimiento Colaborativo de Servicios
(Berant <i>et al.</i> , 2012)	Análisis semántico	Paradigma Vinculación textual
Berton <i>et al.</i> (2015)	Descubrimiento relaciones	Predicción de Enlaces
Chao y Meng-ting (2010)	Descomposición	Algoritmo para selección de servicios web
Das Sarma <i>et al.</i> (2011)	Descubrimiento	Identificación de relaciones temporales
Fang <i>et al.</i> (2014)	Descubrimiento de relaciones	Relaciones en llamadas HTTP
Munonye (2021)	Flujos	Monitoreo de redes de microservicios
Kirby <i>et al.</i> (2021)	Descomposición	Extracción de microservicios
Li <i>et al.</i> (2021)	Descubrimiento	Redes Convolucionales para descubrimiento de servicios
Liu <i>et al.</i> (2018)	Descubrimiento	Agrupamiento de contexto
Rei y Briscoe (2011)	Descubrimiento relaciones	Vinculación textual
Narock <i>et al.</i> (2014)	Descubrimiento	Uso de ontologías
Oberhauser y Stigler (2018)	Flujos	Orquestación de microservicios
Omer y Schill (2009)	Descubrimiento	Análisis semántico de propiedades de servicios
Singh <i>et al.</i> (2015)	Descubrimiento de relaciones	Minería de reglas de asociación

Tabla 3.3: Trabajos relacionados con grafos aplicados al desarrollo de software

Trabajo	Tema	Propuesta
Álvaro Brandón <i>et al.</i> (2020)	Grafos software	Resolución de anomalías
Hejderup <i>et al.</i> (2018)	Manejo de API	Ecosistema de niveles con dependencias
Ibrahim <i>et al.</i> (2019)	Grafos microservicios	Seguridad de microservicios
Li <i>et al.</i> (2018)	Grafos software	Extracción de sentencias de advertencia
Ma <i>et al.</i> (2018)	Grafos de dependencia	Análisis y prueba de microservicios
Ma <i>et al.</i> (2020)	Grafos software	AutoMAP: diagnósticos automatizados
Meng <i>et al.</i> (2015)	Grafos de dependencia	Grafos de control de flujos
Ni <i>et al.</i> (2020)	Grafos software	Análisis de patrones en red
Pahl <i>et al.</i> (2018)	Descubrimiento de relaciones	Modelado de relaciones de comunicación
Wang y Zhang (2017)	Grafos software	Prueba de integración orientada a aspectos
Wang <i>et al.</i> (2019)	Grafos software	Representación de ecosistema de aplicaciones
Guo <i>et al.</i> (2020)	Grafos microservicios	Diagnóstico de Problemas
Ying (2010)	Grafos microservicios	Modelo de Composición de Servicios
Zhu <i>et al.</i> (2019)	Minería de procesos	Automatización del análisis de registros
Zuo <i>et al.</i> (2021)	Minería de procesos	Visualización de relaciones temporales

CAPÍTULO 4

SOLUCIÓN PROPUESTA

Como se discutió previamente en el Capítulo 1, cada empresa cuenta con diferentes requerimientos de acuerdo a su giro y, por consiguiente, puede ensamblar de distintas maneras los microservicios disponibles. Este incremento en las combinaciones posibles complica la generación de escenarios de prueba por parte de los proveedores de estos servicios. Por lo tanto, se busca descubrir relaciones entre microservicios con el fin último de mejorar la generación de dichos escenarios de prueba.

En el presente capítulo, se describe el modelo de solución propuesto para el descubrimiento de relaciones entre microservicios. Primero, se da una vista general de este modelo y las premisas en las que se basa. Enseguida, se describe el tipo de bitácora a procesar, la cual carece de identificadores de traza, tiene accesos concurrentes por parte de los usuarios y también es masiva. Esta última propiedad nos permite asumir que los patrones de uso diario tenderán a emerger a pesar de las relaciones falsas (espurias) que pudieran generarse por efectos de restricción o concurrencia.

Una vez descrito el tipo de bitácora, se describe la metodología propuesta para su procesamiento, que incluye la extracción y refinamiento de una muestra representativa. Posteriormente, se describen los tipos de redes que son obtenidos a través de la extensión de los conceptos de *grafo de secuencia* y *grafo de dependencia*

vistos en el Capítulo 2. Dicha extensión o *relajación* consiste en considerar vecindades de órdenes superiores (“saltos”), y se lleva a cabo para resaltar las relaciones auténticas—debilitando con ello las relaciones espurias.

4.1 DESCRIPCIÓN GENERAL DEL MODELO DE SOLUCIÓN

El modelo de solución propuesto se basa en los siguientes puntos:

- ◇ Es posible realizar descubrimiento de relaciones entre microservicios mediante minería de procesos, específicamente recurriendo a la tarea de descubrimiento de proceso.
- ◇ En este caso, no son de interés los procesos en sí—ya que estos son variados y dependen de las características de los diferentes consumidores de los microservicios. Más bien interesan las posibles secuencias de invocación de microservicios, pues estas ayudarían a obtener escenarios de prueba.
- ◇ Una red de microservicios representaría de manera natural las relaciones entre diferentes servicios. La estructura matemática indicada, por tanto, sería la de un grafo.
- ◇ De entre los algoritmos para minería de procesos, el minero heurístico (Weijters *et al.*, 2006) incluye mecanismos para la generación de grafos; en particular, es posible generar grafos de secuencia y de dependencia. Por lo tanto, sería útil para nuestros propósitos.
- ◇ Debido a que el minero heurístico requiere una bitácora con identificadores de traza para generar grafos de secuencia y dependencia, se necesita acoplar el algoritmo y/o la bitácora para sortear este problema.

- ◇ Para poder trabajar con una bitácora sin identificadores de traza, se propone tratarla como si consistiera en una sola traza—es decir, como si fuera un meta-proceso con un meta-usuario.
- ◇ Como, además, esta bitácora es masiva y concurrente, se propone asimismo un *enfoque relajado* para construir los grafos de secuencia y dependencia. A través de la relajación de estos grafos, es posible obtener una red de microservicios que refleje de manera más precisa las secuencias de invocación.

Para resumir nuestra propuesta de solución, se podría indicar lo siguiente. Viendo las características de la bitácora de microservicios, se extrae la relación entre microservicios a través de un grafo de dependencia relajado, el cual no solamente considera la precedencia inmediata, sino también relaciones de precedencia en general, donde estas relaciones pueden estar separadas por una cantidad h de saltos. La metodología empleada consiste en tres etapas principales: (1) extracción de la bitácora, (2) pre-procesamiento de la bitácora y (3) generación del grafo de dependencia relajado. A continuación, se explican estos tres pasos.

4.2 BITÁCORAS SIMPLES DE MICROSERVICIOS

La bitácora de nuestro caso de estudio corresponde a una *bitácora simple de microservicios*. Para caracterizar este tipo de bitácora, primeramente tendremos en cuenta que cada evento de esta bitácora consiste en la invocación a un microservicio. La forma en que se registran estos eventos depende de la empresa—es decir, no existe un estándar de los atributos a registrar para cada invocación realizada. Por ejemplo, algunas bitácoras pueden contener la URL del microservicio, la firma de tiempo, el usuario que hizo la invocación y la IP desde donde se realizó esta invocación; otras, mientras tanto, pueden contener solamente la URL del servicio y la firma de tiempo. En nuestro caso, consideramos que tenemos una *bitácora simple*, la cual se distingue por ser:

- ◇ *Restringida*. Carece de identificadores de traza.
- ◇ *Concurrente*. Registra eventos simultáneos ocurridos por la presencia de diversos usuarios.
- ◇ *Masiva*. Tiene registrada una cantidad considerable de eventos.

Una *bitácora restringida* se caracteriza por la ausencia de identificadores de traza para los eventos, lo que dificulta, en la práctica, su inferencia. Formalmente, esta propiedad tiene un valor nulo, es decir, $\#_x(e) = \perp$. Como se expuso en el Capítulo 3, existen técnicas para deducir estos identificadores y así poder emplear algoritmos de minería de procesos. Sin embargo, dichos algoritmos de inferencia dependen de estructuras de datos que no siempre están disponibles, como el modelo de proceso “en papel” o información sensible del usuario, como su dirección IP.

Otras varias cuestiones obstaculizan el acceso a los identificadores de traza—entre ellas, las políticas de seguridad de la empresa, el formato original de la bitácora (que puede carecer de información) o simplemente la falta de estructuras particulares en la empresa. En nuestro caso de estudio, la bitácora a la que se tiene acceso solo incluye, para cada evento, la actividad (es decir, la invocación a un microservicio) y la firma de tiempo correspondiente a dicha invocación.

Un desafío más para la inferencia de los identificadores de traza es la concurrencia, ya que puede generar datos espurios. En las *bitácoras concurrentes*, se tienen eventos generados de manera simultánea por diversos usuarios. Dichos usuarios pueden, por ejemplo, encontrarse en diferentes continentes o en diferentes husos horarios. De igual manera, podrían estar utilizando diferentes funcionalidades del sistema de acuerdo a distintas necesidades, por ejemplo, revisar un catálogo de productos o darle seguimiento a una orden. Esto provoca que los eventos de diferentes procesos queden entrelazados en la bitácora, y que la secuencia entre una traza y otra se vea alterada. La Tabla 4.1 ejemplifica este fenómeno.

Las bitácoras con un alto grado de concurrencia también suelen ser masivas.

Tabla 4.1: Ejemplo de una bitácora concurrente

Usuario A	Usuario B	Bitácora
microservicio a	microservicio x	microservicio a 10:00:00
microservicio b	microservicio y	microservicio x 10:00:03
microservicio c	microservicio z	microservicio y 10:01:05
		microservicio b 10:01:06
		microservicio c 10:03:01
		microservicio z 10:05:00

Una *bitácora masiva* se caracteriza por contar con una cantidad significativa de eventos. Considerando que la palabra “significativa” es subjetiva, definimos un criterio para considerar que una bitácora es masiva. Este criterio consiste en comparar su tamaño con el de bitácoras de referencia (tipo *benchmark*). Por “tamaño”, entiéndase la cantidad de eventos que contiene y la cantidad de actividades posibles (i.e., $|A|$); en cuanto a la cantidad de eventos, estamos tomando una muestra estática de la bitácora que ha sido descargada en un lapso de tiempo desde el ambiente de producción.

Viendo lo anterior, consideramos que una bitácora es masiva cuando su tamaño es comparable o sobrepasa el tamaño promedio de bitácoras de referencia; de acuerdo con la literatura (Bayomie *et al.*, 2019), estas bitácoras contienen cientos de miles de eventos ($\approx 290,000$ en promedio, donde la bitácora de mayor tamaño tiene ≈ 1.5 millones), y contienen decenas de actividades (≈ 40). Como veremos más adelante, nuestro caso de estudio supera ampliamente este tamaño. Por lo tanto, podría categorizarse como una bitácora masiva.

4.3 PROCESAMIENTO DE UNA BITÁCORA SIMPLE DE MICROSERVICIOS

Para el procesamiento de la bitácora, se consideran dos aspectos: (a) lógico y (b) físico. El aspecto lógico, por una parte, se refiere a la conceptualización de la bitácora. El aspecto físico, por otra parte, se refiere al tratamiento dado al archivo para poder generar un grafo de dependencia. Por lo tanto, el procesamiento lógico indica cómo se manejará la bitácora para poder darse como entrada a algoritmos de minería de procesos. Mientras tanto, el procesamiento físico indica qué información de la bitácora será recolectada y en qué formato se dejará esta información.

4.3.1 PROCESAMIENTO LÓGICO

Debido a que se cuenta con una bitácora simple de microservicios, es necesario abordar la falta de identificadores de traza. Para este fin, consideramos que la bitácora consiste en una sola traza (“meta-traza”), y que se produce bajo un ambiente de un solo usuario (“meta-usuario”). Por tanto, podemos ver la bitácora como un *meta-proceso*, el cual constantemente se ejecuta a lo largo de los ciclos de negocio. A continuación, exploramos la factibilidad de esta suposición.

La suposición de un meta-proceso es factible por varias razones. Por un lado, los procesos ejecutados conforman ciclos de negocio—siendo, por tanto, repetitivos. Adicionalmente, los procesos críticos de una compañía (e.g., la considerada en el caso de estudio) son típicamente similares entre sí; por ejemplo, colocar una orden y darle seguimiento a una orden comparten actividades como cargar los datos del cliente.

Otra razón para procesar la bitácora como un meta-proceso radica en el incremento de la cantidad de eventos en la meta-traza. Aunque lo anterior puede resultar

en un aumento de relaciones espurias, también podemos asumir que la cantidad de relaciones auténticas será considerablemente mayor. Este mismo principio se aplica a los procesos afectados por la concurrencia, debido a que la probabilidad de que ocurra una relación espuria particular debería ser menor que la probabilidad de que ocurra una relación auténtica—producida con mayor frecuencia a lo largo de los ciclos de negocio.

Visto desde otra perspectiva, podríamos decir que la bitácora simple está sujeta a interferencias o “ruido”, y una forma de disminuirlo es generando una meta-traza. Por tanto, al considerar la bitácora como una sola secuencia de eventos, estamos presuponiendo que las relaciones auténticas surgirán a pesar de este ruido. Dado que los procesos tienden a repetirse, se espera que los patrones cotidianos en la compañía puedan destacarse y reflejarse en el grafo de dependencia. Este enfoque también ayudaría a mitigar la falta de identificadores de traza.

4.3.2 PROCESAMIENTO FÍSICO

El procesamiento físico de la bitácora se divide en dos tareas: (a) *extracción* de una muestra representativa de la bitácora y (b) *refinamiento* de la muestra obtenida. La primer tarea es necesaria debido a que, en el ambiente de producción, las bitácoras son altamente dinámicas y están constantemente registrando los eventos del día a día; por tanto, para realizar un análisis es necesario descargar una versión estática que represente los ciclos de negocio. La segunda tarea es necesaria para facilitar la construcción del grafo de dependencia.

4.3.2.1 EXTRACCIÓN

La tarea de extracción consiste en recolectar una muestra representativa de la bitácora de eventos, y conlleva diferentes decisiones. El resultado es una *bitácora*

en bruto (denotada como \mathcal{L}_b), que puede ser posteriormente refinada para distintos fines. Las variables que se toman en cuenta para la extracción pueden dividirse en *independientes* y *dependientes*. En este caso, las variables independientes indican (a) qué es lo que se quiere lograr y (b) qué es lo que se tiene para lograr este objetivo. Con base en las variables independientes, las variables dependientes indicarían el *qué*, *cuándo* y *cómo* de la extracción. Definir los valores de estas variables, a su vez, resulta en un tamaño de muestra.

Como variables independientes, se toman:

1. Variables independientes

- a) Objetivo de la extracción
- b) Medios para la extracción
 - 1) Formato de la bitácora
 - 2) Herramientas
 - 3) Restricciones

2. Variables dependientes

- a) Rango de fechas a extraer
- b) Modo de extracción
- c) Tiempo para la extracción
 - 1) Fechas
 - 2) Horarios
 - 3) Duración

En cuanto a las variables independientes, el objetivo de la extracción orienta el proceso de recolección de la muestra; posibles objetivos incluyen el representar ciclos completos de negocio, o el poder recolectar en la muestra todo el universo de actividades. Los medios de extracción, por otra parte, incluyen el formato de la

bitácora, las herramientas disponibles y otras restricciones, donde estos, respectivamente indican la información disponible en los registros, las capacidades de software y hardware, y las dificultades que constriñen la recolección de la muestra (e.g. políticas de seguridad de la compañía).

En cuanto a las variables dependientes, el rango de fechas a extraer se refiere a las fechas que tendrán los registros a recolectar (mismas que deberían estar alineadas con el objetivo). El modo de extracción, por otro lado, se refiere a definir si la muestra se recolecta en tiempo real (durante la operación diaria) o si se recolecta de manera diferida (con los eventos que ya están registrados una vez que terminó el horario de operación). En cuanto al tiempo para la extracción, se refiere a las fechas y horarios en que se hará la extracción, y cuánto durará cada sesión de recolección de registros. Tanto el modo como el tiempo de extracción están supeditados a las herramientas y restricciones, como el tamaño de la compañía y la existencia de diferentes husos horarios dentro de esta; en una compañía multi-nacional, por ejemplo, se supone que habrá usuarios utilizando los sistemas a lo largo del día.

Vale la pena mencionar que, a pesar de estarnos refiriendo a la muestra extraída de la bitácora, continuaremos utilizando solamente el término *bitácora* para facilitar la comprensión.

4.3.2.2 REFINAMIENTO

Para obtener una bitácora utilizable, es necesario refinar la bitácora en bruto. Por lo tanto, se realiza un pre-procesamiento sobre \mathcal{L}_b , de tal manera que solamente nos quedemos con una representación adecuada del evento; en este caso, se ocuparía solamente la actividad y la firma de tiempo correspondiente a esta actividad. Esto incluye operaciones como el filtrado de información relevante, y la transformación de las llamadas a servicio a identificadores de actividad, así como de la firma de tiempo a pasos de tiempo, por ejemplo. Una vez que la bitácora está refinada, terminamos

con una bitácora simple de microservicios \mathcal{L}_p que ya se puede emplear como entrada para generar grafos relajados de secuencia y dependencia.

4.4 DESCUBRIMIENTO DE RELACIONES ENTRE MICROSERVICIOS

Para descubrir relaciones entre microservicios, se plantea generar un grafo de dependencia entre las diferentes actividades de la bitácora. En ese sentido, el grafo de dependencia representa una red de precedencias entre servicios —es decir, un grafo donde es posible visualizar qué servicio se invoca antes de qué otro servicio y qué tan frecuente es esta precedencia. Para generar este grafo de dependencia, se requiere primero la generación de un grafo de secuencia. Además, como parte de nuestra contribución, se generan versiones *relajadas* de ambos grafos, donde las precedencias de los servicios no son estrictamente inmediatas, sino que pueden tener invocaciones intermedias. Esto se hace para reducir el impacto de las limitaciones y el acceso simultáneo al registro.

4.4.1 GRAFO DE SECUENCIA RELAJADO

En nuestro caso concreto, proponemos una versión relajada del grafo de dependencia que también considera las relaciones de precedencia *indirecta*. Este grafo de dependencia relajado se propone debido a la naturaleza restringida de la bitácora de eventos, que no está etiquetada a ningún proceso de negocio —es decir, carece de identificadores de traza. También, como hemos visto en la Sección 4.2, la bitácora es concurrente al estar siendo escrita por varios usuarios al mismo tiempo; lo anterior ocasiona que las llamadas consecutivas se dispersen. Por lo tanto, relajar la restricción de precedencia inmediata y permitir una ventana de eventos intermedios entre un par de llamadas a microservicio podría reducir esta dispersión.

Para crear una versión relajada del grafo de dependencia, primero debemos relajar también la definición de medida de secuencia. Para hacer esto, adaptamos el concepto de vecinos de *saltos* de la teoría de grafos (Sección 2.2) a un *nivel de relajación*, el cual denotamos con h .

El nivel de relajación h puede definirse como la *cantidad máxima de saltos que puede haber entre dos vértices para considerar que son vecinos*. Visto de otra manera, consideremos un camino P cuyo origen es el vértice s y cuyo destino es el vértice t , de tal suerte que $P = \langle s, \dots, t \rangle$. En este caso, el nivel de relajación representaría la longitud máxima de este camino:

$$h = |P|_{\max}, \quad (4.1)$$

donde utilizamos $|\cdot|_{\max}$ para representar la longitud máxima de un camino entre un par de vértices en un grafo.

Por ejemplo, si $h = 1$, entonces solamente puede haber un salto (conexión) para llegar de s a t . En otras palabras, solamente se considera que s y t son vecinos si existe precedencia inmediata entre ellos. Sin embargo, si $h = 2$, entonces ahora se consideraría que s y t son vecinos tanto por precedencia inmediata como por *precedencia indirecta* con a lo más un nodo (evento) intermedio (o dos saltos). De esta manera, tanto las ocurrencias de $\langle s, t \rangle$ como las ocurrencias de $\langle s, e, t \rangle$, donde $e \in \mathcal{E}$, serían tomadas en cuenta para calcular la medida de secuencia $|s \succ_{\mathcal{L}} t|$. De manera semejante, si $h = 3$, se tomarían en cuenta los dos tipos de secuencia anteriores y además $\langle s, e, f, t \rangle$, donde $e, f \in \mathcal{E}$. En general, cuando $h = k$ existe una cantidad máxima de k saltos entre s y t .

Otra manera de definir h es mediante sub-secuencias. Consideremos una secuencia de eventos $S = \langle s, S', t \rangle$, donde $s, t \in \mathcal{E}$ y S' es una sub-secuencia de S . En este caso, l sería la longitud máxima de S' , de tal suerte que cuando $l = 0$, s precede inmediatamente a t . De modo semejante, cuando $l = 1$, existe a lo más un evento intermedio entre s y t , y cuando $l = 2$, existen a lo más dos eventos intermedios entre s y t . Entonces, $h = l + 1$.

También es importante considerar que, entre mayor es la distancia entre los eventos s y t , la fuerza de esta relación de precedencia suele ser más débil, ya que comienza a ser remota. Es decir, también va bajando la probabilidad de que la relación entre s y t sea una relación auténtica (y no espuria). Por esta razón, manejamos una contribución degradada c que representa la fuerza de la relación que va decayendo al tener eventos intermedios. En este caso, consideramos que $c = 1/h$, ya que nos parece una selección apropiada para una primera versión del método propuesto.

Viendo lo anterior, de los tres atributos que toma el minero heurístico (identificador de traza, actividad y firma de tiempo), solamente se toman los dos que están disponibles (actividad y firma de tiempo). Formalmente, $\forall e \in \mathcal{E} : e = [a, t]$. De igual manera, se toma la bitácora como una sola traza y, en consecuencia, $\mathcal{L} = \sigma$ y $|\mathcal{L}| = 1$, asumiendo que lo anterior nos permitirá descubrir patrones de repetición esperados por la naturaleza masiva de la bitácora. De esta manera, la medida de secuencia relajada está dada por la Definición 4:

Definición 4 *La medida de secuencia relajada es la cantidad de veces que el evento s precede al evento t con una distancia h dentro de la bitácora \mathcal{L} .*

Teniendo en cuenta el nivel de relajación h (“saltos de evento” máximos) de s a t , así como la aportación de $c = \frac{1}{h}$, la medida de secuencia relajada describe las condiciones anteriores:

$$|s >_{\mathcal{L}} t|_h = \sum_{k=1}^{k=h} \frac{1}{k} \cdot |\{\pi(s, t) : \sigma(i) = s \wedge \sigma(i+k) = t\}| \quad (4.2)$$

donde establecemos que:

- $i > 0$ y $i < |\sigma| - 1$

La definición de un grafo de secuencia relajado se basa, entonces, en la medida de secuencia relajada dada por la Ecuación 4.2. En consecuencia, cada peso $w(s, t)$ está dado por

$$w(s, t) = |s >_{\mathcal{L}} t|_h. \quad (4.3)$$

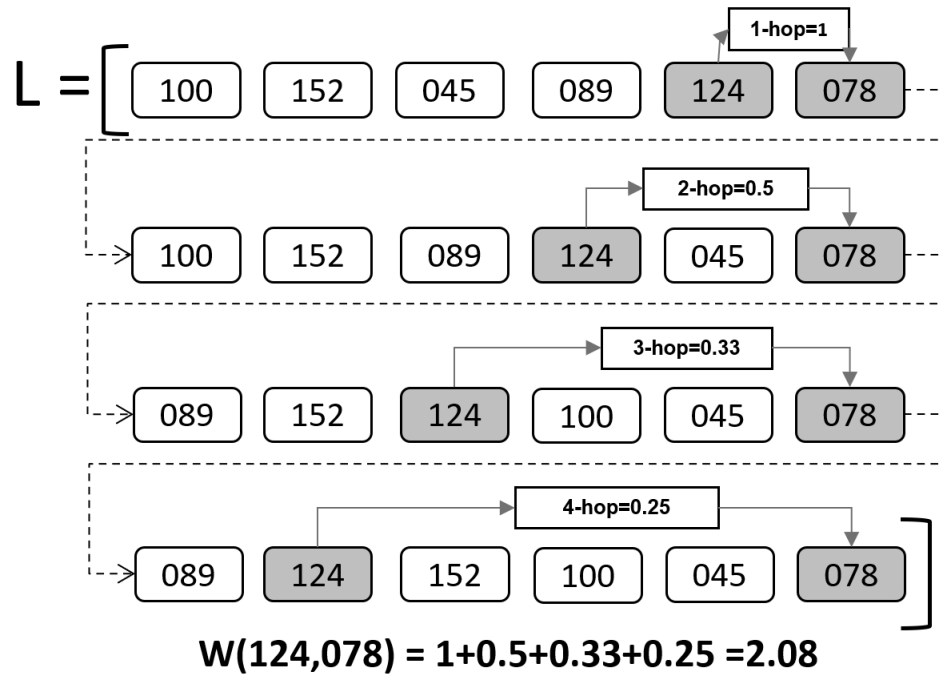


Figura 4.1: Ejemplo del cálculo de $w(a, b)$ con $h = 4$ (Eq. 4.3)

Como podemos ver, esta definición es análoga a la de la medida de secuencia denotada en la Ecuación 2.2 (Capítulo 2, página 24), con la diferencia de los cálculos en los pesos de cada conexión.

A manera de ejemplo, en la Figura 4.1, $w(a, b)$ se calcula para eventos a y b que corresponden a las actividades 124 y 078, respectivamente. Cuanto más alejados estén estos eventos uno del otro, menor será $w(a, b)$, teniendo $c = 1/1 = 1$ para $h = 1$ (un salto) y $c = 0.25$ para $h = 4$ (cuatro saltos). De hecho, si consideramos $h = 4$, tenemos una ocurrencia de precedencia inmediata, una ocurrencia con un evento intermedio, una ocurrencia con dos eventos intermedios y una ocurrencia con tres eventos intermedios. Por tanto, $w(a, b) = 1 + 0.5 + 0.33 + 0.25 = 2.08$.

4.4.2 GRAFO DE DEPENDENCIA RELAJADO

De manera similar a la Ecuación 2.3, la *medida de dependencia relajada* tiene en cuenta la medida de secuencia relajada expresada en la Ecuación 4.4:

$$|s \Rightarrow_{\mathcal{L}} t|_h = \begin{cases} \frac{|s >_{\mathcal{L}} s|_h}{|s >_{\mathcal{L}} s|_h + 1} & \text{si } s = t \\ \frac{|s >_{\mathcal{L}} t|_h - |s >_{\mathcal{L}} t|_h}{|s >_{\mathcal{L}} t|_h + |s >_{\mathcal{L}} t|_h + 1} & \text{si } s \neq t \end{cases} \quad (4.4)$$

4.5 RESUMEN DEL CAPÍTULO

Este capítulo describe la solución propuesta, que consiste en la extracción de un grafo de dependencia relajado a partir de una bitácora de eventos restringida, masiva y concurrente. En este caso, el grafo de dependencia relajado es una generalización del grafo de dependencia generado por el minero heurístico (Weijters *et al.*, 2006). Una bitácora de eventos restringida no cuenta con identificadores de traza que pudieran permitir dividirla en una secuencia de trazas, y tampoco permite la inferencia de dichos identificadores. Para abordar esta dificultad, se asumió que la bitácora en sí misma constituye una sola traza.

Tomando en cuenta lo anterior, se introdujo el nivel de relajación h para permitir relaciones de precedencia indirectas, y con ello mitigar el problema de la concurrencia. Este nivel se define como la cantidad máxima de saltos que pueden existir entre dos eventos para considerarlos vecinos (es decir, que pertenecen a una relación de precedencia). De igual manera, h permitió definir de manera relajada las medidas de secuencia y dependencia, y así proponer la versión relajada de los grafos respectivos.

CAPÍTULO 5

EXPERIMENTOS Y RESULTADOS

Este capítulo presenta los ensayos realizados para verificar la hipótesis propuesta. Estos experimentos se dividen en dos grupos: pruebas de cobertura y pruebas de descubrimiento. Por una parte, en las pruebas de cobertura se valida el grado en que el grafo de dependencia refleja diferentes grafos de referencia (*ground truth*). Por otra parte, en las pruebas de descubrimiento se valida que las relaciones nuevas obtenidas a partir de este grafo sean significativas y no meramente ruido.

Viendo lo anterior, en las pruebas de cobertura se utilizan varios niveles de relajamiento y se comparan estos resultados contra el nivel base sin relajación ($h = 1$). Mientras tanto, en las pruebas de descubrimiento, se contrasta el grafo contra un modelo de configuración, que es un modelo de grafo aleatorio donde se respeta la secuencia de grado (ver Sección 2.2.1). El análisis estadístico que complementa los resultados puede consultarse vía QR (Figura A.1). Se irá, asimismo, haciendo mención de puntos particulares del apéndice a medida que se requiera.

5.1 PREPARACIÓN DE LOS DATOS

El caso de estudio de nuestra investigación está basado en el sistema de comercio electrónico para la compañía de Cementos Mexicanos (CEMEX). La arquitectura

de dicho sistema está basada en múltiples capas tecnológicas—entre ellas, una basada en servicios web y en microservicios. La bitácora de registro de microservicios es obtenida de una plataforma basada en Microsoft Azure (Microsoft, 2022a) y representa la actividad del sistema de comercio electrónico implementado en más de una veintena de países alrededor del mundo. También se registran en esta bitácora las llamadas ejecutadas sobre docenas de procesos de negocios implementados vía interfaces usuario (Web y móvil).

5.1.1 EXTRACCIÓN

La primera fase de nuestra metodología indica la recolección de una muestra representativa de la bitácora, de tal forma que nos permita una manipulación eficiente y repetible sobre los experimentos diseñados. El resumen de la extracción puede verse en la Tabla 5.1.

Tabla 5.1: Resumen de la extracción

Variable	Valor
<i>Objetivo</i>	Capturar al menos un ciclo de negocios
<i>Medios: formato</i>	URL de invocación a microservicio
<i>Medios: herramientas</i>	API de Azure, KQL
<i>Medios: restricciones</i>	Bloqueo de información por seguridad, límite de 10 000 resultados por consulta KQL
<i>Rango de fechas</i>	71 días hábiles
<i>Modo</i>	Mixto: durante operación, pero registros ya existentes
<i>Tiempo: fechas</i>	Septiembre 2019, noviembre 2020
<i>Tiempo: horarios</i>	9am-10am CST
<i>Tiempo: duración</i>	7 días

OBJETIVO. El objetivo de la extracción fue capturar un ciclo de negocios, pues se supone que durante este ciclo se llevarán a cabo todas (o la mayor parte) de las actividades. Si bien la mayoría de los ciclos comerciales se completan en un mes de operación, también es cierto que varios ciclos confirmarían los datos observados en un solo ciclo. En ese sentido, se consideró extraer los registros de los últimos tres meses—considerando, además, que en la bitácora de nuestro caso de estudio sólo se almacenan los últimos tres meses de operaciones. Asimismo, ya que extraer tres meses de operaciones nos arrojaría aproximadamente 400 millones de registros de la bitácora, se consideró que este volumen de datos nos permitía poder realizar experimentación.

FORMATO DE LA BITÁCORA. Cada registro que se encuentra en la bitácora de eventos (\mathcal{L}) tiene varias columnas de datos. Sin embargo, las únicas columnas que se tomaron para esta investigación por su relevancia fueron la firma de tiempo y la llamada de URL del microservicio, compuesta por método HTTP, versión de API, nombre de API, operación de API y atributos adicionales relacionados con la llamada del microservicio. El modelo RESTful tiene características que permiten estandarizar su uso, tales como el tipo de operación HTTP para determinar el comportamiento del método. Por ejemplo, dado un método `http` de tipo `post`, tenemos una URL de tipo

`http://server/V1/IDM/CredencialesUsuario`

Para esta llamada, podemos decir que se invoca para escritura la operación `CredencialesUsuario` a través de la API `IDM` con la versión `V1`.

HERRAMIENTAS. En cuanto a las herramientas, el ambiente de producción de la compañía cuenta con plataformas y lenguajes a través de los cuales se puede acceder a la bitácora. Este acceso se puede hacer de varias maneras, ya sea a través de la interfaz de usuario de *Azure portal* o a través de código usando el *API* de

```
let start=datetime("2019-08-1T00:00:00.000Z"); let end=datetime("2019-08-1T00:00:59.999Z"); let
dataset=requests| where timestamp > start and timestamp < end| where client Type!= "Browser"
;dataset
```

Figura 5.1: Ejemplo de KQL

Azure. Utilizando este API, se encontró una estructura de tablas en el servidor de monitoreo, de la cual era posible obtener la bitácora de eventos. Esta tabla contiene la información requerida en la llamada `request`. El lenguaje de consulta específico para esta plataforma es el *Lenguaje de búsqueda Kusto*, también conocido como KQL (Microsoft, 2022b). La Figura 5.1 presenta una consulta en este lenguaje.

Por medio de múltiples consultas a las tablas disponibles, se observó que la estructura propuesta por KQL satisfacía la información requerida. A pesar de que la interfaz donde se ejecutaron las consultas de KQL soportaba consultas más elaboradas, solamente se buscó obtener datos almacenados en la tabla *Request*, dado que necesitábamos un volumen de datos considerable para probar. Sin embargo, debido a que la plataforma no fue inicialmente diseñada para este tipo de consultas, fue necesario diseñar también otros mecanismos para la extracción.

RESTRICCIONES. Dentro de los registros arrojados por las consultas, hubo campos que no fueron utilizables. Por ejemplo, el campo de `Client_IP` tenía ceros como valor (0.0.0.0). Lo anterior fue debido a políticas de seguridad de la compañía. En este caso, la compañía utiliza una herramienta que cumple con la función de balancear la carga y administrar el caché y la seguridad de los usuarios (Imperva, 2022). A pesar de que dicha aplicación maneja de manera eficiente los servidores web, se bloquea cierta información como efecto de protección a los datos de usuario. Otra limitación en el acceso a la bitácora se dio por el uso de la interfaz gráfica de *Azure*, ya que se pueden obtener hasta máximo 10,000 registros por consulta. Lo anterior, debido a que la principal función de este módulo es el análisis por parte de los equipos de operación. Por tanto, se tuvo que trabajar bajo estas restricciones.

RANGO DE FECHAS A EXTRAER. Debido a que se deseaba extraer varios ciclos de negocio, se obtuvieron los registros correspondientes a 71 días hábiles. Estos días correspondieron a las fechas del 10-31 de agosto de 2019 (21 días), del 1°-30 de septiembre de 2019 (30 días) y del 10-19 de octubre de 2020 (20 días).

MODO DE EXTRACCIÓN. Dadas las restricciones en las consultas de KQL, se generaron consultas de manera sistemática para poder extraer la muestra de la bitácora. En ese sentido, se diseñó una serie de consultas estáticas (es decir, que codificamos previamente), donde cada consulta obtenía todos los registros de eventos ocurridos en el intervalo de un minuto. Puesto que en un minuto suceden aproximadamente 7,000 eventos, este intervalo fue escogido para no sobrepasar el límite de los 10,000 registros permitidos. Una de estas consultas podía extraer, por ejemplo, todos los eventos ocurridos del 22/09/2019 11:59:00 al 22/09/2019 11:59:59. Estas consultas estáticas fueron aproximadamente 130,000.

Las consultas estáticas en KQL se realizaron por medio un script en Python (Python, 2022), que generaba cada consulta y la almacenaba en un archivo de texto. También se desarrolló una herramienta en Visual Studio 2019 para acceder vía API a *Microsoft Application Insight*. Para este fin, se gestionó una llave de acceso remoto vía API en la plataforma. Como resultado, la herramienta toma las consultas predefinidas por el intervalo de tiempo definido y genera un archivo de texto por cada consulta ejecutada exitosamente. Un ejemplo del contenido de una consulta se muestra en la Figura 5.2.

En cuanto a la forma de hacer la extracción, se puede decir que fue *mixta*, ya que se hizo durante las jornadas de operación de la compañía, pero tomando registros que ya existían previamente en la bitácora. Es decir, a partir de la fecha T de extracción, se hicieron consultas para extraer x intervalos de un minuto, yendo hacia atrás desde T hacia el registro más antiguo de la bitácora.

Para determinar el rango de tiempo máximo que nos podía garantizar no perder

2020-01-24T16:11:11.7387544Z
d47d08bfc8a54cbaa7197714be71a937.33ec106ad043461b_
GET /v7/dm/jobsites/summary
https://api.cemexgo.com/v7/dm/jobsites/summary?dateTo=2020-01-26T06:59:59&orderBy=programedDateTime&dateFrom=2020-01-25T07:00:00
True
200
<250ms
request
7714be71a937", "Operation Name": "getJobsiteSum", "Product Name": "5a85d081c84a9a09901f137f", "HTTP Method": "GET", "Response-Body": "{ \"jobsites\": [{ \"jobsiteId\": 934158, \"jobsiteCode\": \"0066838661\", \"jobsiteDesc\": \"PAVLLVM Rafael Buelna - JAPAVA\", \"isJobsiteActive\": false, \"orders\": [{ \"orderId\": 14913051, \"orderStatus\": \"No Inicializado\", \"orderStatusGroupCode\": \"NST\", \"isReadyMix\": true, \"totalQuantity\": 70.0, \"deliveredQuantity\": 0.0, \"totalLoads\": 10.0, \"totalDeliveries\": 0.0, \"programedDateTime\": \"2020-01-25T12:00:00\", \"favoriteStatus\": \"N\", \"longitude\": -106.433555, \"latitude\": 23.246812, \"altitude\": 0.0 }] }, { \"API Name\": \"contactcall_7.0.0\", \"Cache\": \"None\", \"Request-App-Code\": \"Foreman_App\", \"Request-User-Agent\": \"okhttp/3.12.0\", \"Request-X-IBM-Client-Id\": \"1d40a262-d6c6-47fb-b2ec-e54ad233b114\", \"Request-Host\": \"api.cemexgo.com\", \"Request-Jwt\": \"eyJlbmMiOiJBMjU2Q0JDLUhNTnEYliwiYWxnIjoieUINBLU9BRVAtMjU2liwiY3R5IjoieUln0u.voGlna94umWhlJlPosa9JP7LpnTPlqk--8K14opLd85b-cNuh0UYgabuVpqVkuSUE5T4oifzxwR8SAhPF5TAijDvBzGOpQ1X4R-RbD1Av_4lnQwgo7MSJVbDWATeQylcT6c8oqcLgZWS8XGiaPw1Q39gpQn-nbEuXmPl8-96_gfwDbNmGPxAYi4020bdxgkzZA84dL6U7U309VpmM4w6WzFUo8vjCEOeX_xWsbXvpZYKyMDJlZyqZi5ldWZ88741giCo_0-XbqRPn2H_sUpt5doJeX7pMjxunPcw4HlotblwvdE2m6dwJ6ueaiZpSV89nwf1LdReHZU9I9I58JQ_j15udrWzkOECOFhH3XAU1g.1vul2M811K9-_rwkruYxn2JaseKV9KB67J11VQaXvwKvTp7XUzZTkO53A9kpX3QKrB4NPpFVCRP7zpp8UqCeJliepe_OtsjOztXPfhtAZ58yEqFTPNpEbSJHjdqxl_5IzJakYcppe6VKKwZjZo6WVeWXgSSgs_-6IazEeqX1HtmqXoj9zqSgijvGNsl0Zp_G3T7i9q2SQInDs4RdxJE4VRQZKJE_KAIhfujZZ4JLEHUt8WyywnDcpM6xJT7jMCuk7PXNUVIS9XbKpoyDch743OP2nqY0CyQv4HHUjFtUtonyJZezm5QnnTXB3dWZ4xtsazD39zsGUr17HvD2v9eHDRY6zhYWHLR9I0Pxx8dGYglOMTstplw5jSW6amgunRb8cZ { \"Client Time (in ms)\": 0, \"Response Size\": 617, \"Request Size\": 3554 }
GET /v7/dm/jobsites/summary
d47d08bfc8a54cbaa7197714be71a937
d47d08bfc8a54cbaa7197714be71a937
PC
0.0.0.0
Texas
United States
cemex South Central US
cemex South Central US
8a08ff00-801a-4f31-9a6b-6293d69cea3a
uscldcnxiapimgmtp01
a3f7a379-3faa-41fe-b033-268d9c66c320
apim:0.12.885.0
24268e5d-3ec4-11ea-8783-95749743fd35

Figura 5.2: Detalle de las consultas

información y otra forma de acceder a esta plataforma, para el primer punto, se eligió ejecutar una variedad de rangos de tiempo en horas de máxima operación esperada, es decir cuando los países con mayor operación de usuarios estuviesen trabajando. Se encontró que los países con mayor carga se encontraba en los países de México, Colombia, Reino Unido, y España.

TIEMPO PARA LA EXTRACCIÓN. Debido a los husos horarios de los países antes mencionados, la extracción se hizo de 9 am-10 am CST, que es cuando se esperaba que estos cuatro países estuviesen más activos. El proceso de extracción duró 7 días.

Al finalizar el ejercicio de extracción de la bitácora, se obtuvo un volumen

de archivos cuyo tamaño en conjunto superaba los 500 GB de espacio. Estos datos constituyen la bitácora en bruto \mathcal{L}_b y, por ende, fueron la materia prima para diseñar y probar el método propuesto para descubrimiento de relaciones entre microservicios.

Se tomó la decisión de no eliminar información de la bitácora en bruto, pues el esfuerzo de obtenerla fue alto—tanto en términos de accesos y carga a los servidores, como en términos de diseñar las consultas para evitar recibir penalización por parte del equipo de seguridad. Por otra parte, resultó más sencillo hacer el filtrado de información de manera local. Es necesario recalcar que el ambiente donde se obtuvieron estos datos fue productivo. Esta consideración es muy importante para el refinamiento.

5.1.2 REFINAMIENTO

Todos los registros obtenidos en el paso anterior tenían un volumen considerable de espacio y mucha información que no resultaba útil para su procesamiento y, por el contrario, complicaba su entendimiento. Por ejemplo, los datos registrados como parte de la llamada a la URL del microservicio, si bien se referían al mismo método en múltiples ocasiones, también empleaban parámetros distintos en cada ocasión. Para poder identificar cuántas actividades únicas realmente se tenía y definir el universo de las mismas (A), se procesaron cuatro niveles de información:

- **Método HTTP:** este consiste en el tipo de llamada que se está ejecutando, y las opciones disponibles están definidas por la versión de HTTP usada, pero típicamente se encuentran operaciones como: `POST`, `GET`, `PUT`, `HEAD`, `OPTION`, `etc..`
- **Versión:** es descrita por la letra V + un número, tal como $V1, V2 \dots Vn$, de modo tal que puede existir una combinación donde la única diferencia sea la versión.

- **Nombre de la API:** para el caso estudiado, es un acrónimo compuesto de tres a cuatro letras que identifican una capacidad del negocio y que se usa para tener un diccionario de API asociadas contra las capacidades del negocio a nivel de proceso.
- **Método:** es el nombre específico del método de la API que está siendo invocado.

Utilizando la nomenclatura descrita anteriormente, se cargaron todos los registros a una tabla en una base de datos compatible con ANSI SQL (MariaDB, 2022) de manera local. La información almacenada fue el método HTTP, una cadena de texto que representa los niveles de información descritos y la firma de tiempo en el formato `YYYYMMDD-THMMSS.MMMM`. El resto de la información contenida en la URL fue eliminada.

Con la información obtenida, se hizo una identificación de registros únicos compuestos por método HTTP y la URL reducida. Se obtuvieron un total de 978 combinaciones únicas que se repetían en la bitácora. Consideramos que estas combinaciones únicas corresponden al conjunto de *actividades* de la bitácora (explicado en la Sección 2.3.1). Se creó una tabla que se utilizaría como llave, donde a cada actividad se le asignó un identificador numérico. La Figura 5.3 ilustra lo previamente explicado.

Una vez asignados los identificadores, se generó una tabla que representaba a toda la bitácora conteniendo únicamente dos campos: el *identificador de actividad* y la *firma de tiempo*. Aún con esta información simplificada, se ejecutó una reducción adicional. Dado que lo relevante es la secuencia ordenada en el tiempo en que aparecen los eventos en la bitácora, se generó un vector de eventos en base a una consulta ordenada por la firma de tiempo, yendo de los más antiguos a los más nuevos. De este modo, al final sólo se tiene como entrada un vector de valores numéricos que representan actividades (únicas) en secuencia según su aparición en la bitácora original de acuerdo a su firma de tiempo, como muestra el ejemplo de la Figura 5.4.

http	API3	OpKey
GET	v5/ds/documentsbycustomer	5
PATCH	v5/sm/opportunities	12
GET	v5/sm/opportunitybusinesslines	14
GET	v7/dm/jobsites	16
PATCH	v5/sm/opportunitybusinesslines	24
OPTIONS	v5/sm/opportunities	29
OPTIONS	v5/sm/myshipmentlocations	33
OPTIONS	v5/secm/users	42
GET	v6/ce/countries	49
GET	v4/sm/productlines	52
GET	v1/rc/getfiltersbyuser	53
GET	v2/arm/legalentityjobsites	57
GET	v2/arm/accountssummary	59
POST	v2/pe/bulks	60
PATCH	v6/sm/orders	65
POST	v2/secm/oam	67
GET	v2/im/contactpersonroles	74

Figura 5.3: Ejemplo de actividades (únicas) registradas

```
array([605, 605, 245, 245, 245, 488, 67, 67, 67, 67, 67, 605, 245,
605, 276, 245, 146, 605, 245, 276, 276, 602, 156, 161, 508, 155,
605, 245, 326, 605, 245, 215, 488, 146, 488, 276, 396, 49, 146,
605, 245, 276, 618, 447, 110, 103, 326, 67, 67, 341, 605, 605,
326, 276, 245, 103, 75, 104, 508, 245, 75, 605, 75, 605, 245,
245, 605, 75, 245, 49, 605, 276, 245, 75, 146, 488, 396, 605,
245, 488, 75, 276, 49, 146, 488, 396, 605, 488, 245, 605, 618,
605, 326, 245, 245, 146, 618, 605, 245, 326, 605, 605, 605, 245,
245, 605, 605, 245, 245, 605, 245, 245, 67, 605, 245, 605, 245,
605, 245, 605, 605, 605, 245, 245, 245, 605, 605, 605, 245, 245,
245, 605, 245, 605, 245, 605, 245, 67, 57, 215, 273, 605, 605,
49, 276, 488, 146, 245, 276, 618, 618, 215, 234, 146, 215, 157,
234, 605, 220, 245, 59, 384, 384, 57, 248, 605, 605, 110, 447,
540, 245, 605, 245, 245, 605, 245, 448, 146, 49, 488, 276, 234,
215, 215, 157, 234, 146, 220, 57, 67, 384, 384, 59, 605, 245,
605, 245, 276, 605, 605])
```

Figura 5.4: Ejemplo de vector en la bitácora refinada

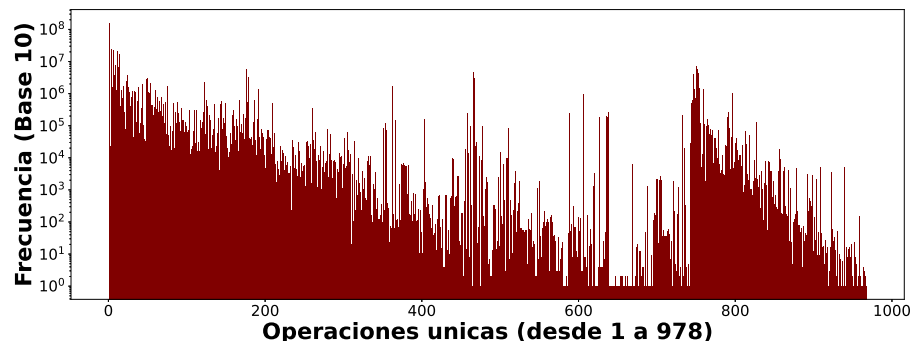


Figura 5.5: Descripción de la frecuencia de registros (escala logarítmica)

Con la información refinada, se ejecutó un análisis de las características de la bitácora. Ya que teníamos 978 actividades, encontramos que la distribución de frecuencias de dichas actividades tenía grandes variaciones. En ese sentido, pocas actividades tenían una frecuencia alta, y muchas otras tenían una frecuencia baja; dentro de este grupo, había otras cuya frecuencia podría considerarse poco relevante, ya que era en extremo baja. En la Figura 5.6, se puede observar estas frecuencias, donde se puede notar cómo hay pocas actividades que llegan a la escala de millones de apariciones, mientras que un grupo amplio cae en la escala de cientos de miles y otros más en los miles. Esto tiene sentido desde la perspectiva de negocio, porque existen microservicios que son altamente usados por múltiples procesos. Por tanto, el valor de esta investigación recae en el descubrimiento de las relaciones producto de esa centralidad.

El conjunto de actividades se registró en un diccionario de tipo (valor numérico, operación del microservicio). Este diccionario tiene dos propósitos clave: mantener un registro de operaciones únicas encontradas en \mathcal{L} , y mantener una relación entre las llaves numéricas y las URLs para poder eventualmente interpretar los resultados.

Otra característica relevante es el nivel de concurrencia con que estas actividades fueron realizadas, ya que la aplicación que se tomó como base para este trabajo se encuentra disponible para operar en 21 países con múltiples usuarios concurrentes. Por esta razón, estudiamos la bitácora refinada para entender cuántas actividades

fueron registradas por lapso de tiempo y cómo es el nivel de concurrencia en términos de eventos registrados por hora tomando en cuenta el horario central (CST).

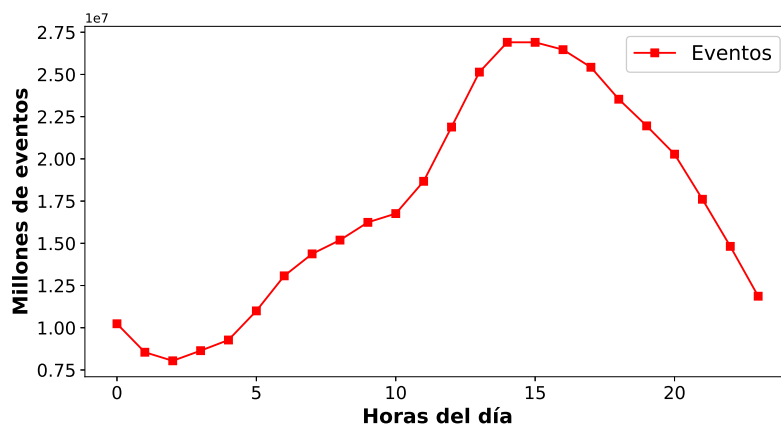
En la Figura 5.6 se puede observar en términos de millones de eventos registrados por día para el rango de fechas contenido en la bitácora extraída. En ese sentido, existen picos de actividad en algunos días, los cuales se pueden explicar por eventos de negocio tales como liberación a ciertos mercados de ciertas funcionalidades. Por lo mismo, lo que vemos es producto de la adopción de estas funcionalidades por parte de los usuarios en ciertas regiones.

Lo relevante de este análisis de la bitácora es el hecho de reconocer que, si bien se tiene un conjunto limitado de actividades, los registros de estas actividades siguen patrones de negocio. Estos patrones, ante la ausencia de un identificador de traza, complementan las firmas de tiempo para poder hacer minería de procesos. Por esta razón, el factor de concurrencia es muy relevante en términos de los resultados que se obtengan; también resulta relevante que nuestro método propuesto está diseñado para trabajar con este factor.

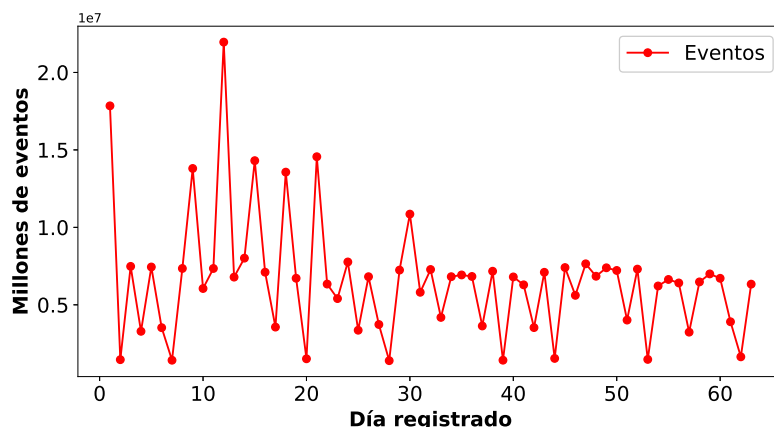
5.2 PRUEBAS DE COBERTURA

Las pruebas de cobertura evalúan qué tan bien representa el grafo de dependencia relajado (con diferentes niveles) a una serie de *grafos de prueba*. Considerando lo anterior, obtuvimos grafos de dependencia con niveles desde $h = 1$ hasta $h = 20$. Las propiedades más sobresaliente de estos grafos pueden encontrarse en la Tabla 5.2.

En cuanto a los grafos de prueba, se trabajó con dos tipos: (a) grafos de referencia y (b) grafos aleatorios. Cada uno de los grafos de referencia contiene relaciones que ya han sido utilizadas por el equipo de pruebas de la compañía. Los grafos aleatorios, por otra parte, se generaron para obtener una perspectiva más clara de los resultados, siguiendo para ello el modelo Erdős-Rényi. Por lo mismo, se espera una



(a) Eventos registrados por hora (millones)



(b) Eventos registrados por día (millones)

Figura 5.6: Descripción de la frecuencia de registros

Tabla 5.2: Propiedades de los grafos de dependencia generados

Nivel (h)	Conexiones	GRADO			
		Promedio	Desv. est.	Mediana	Máximo
1	91863	93.9	96.7	68	476
2	110662	113.2	107.9	89.5	541
3	123589	126.4	115.6	104.5	578
4	131118	134.1	119.1	114	596
5	136937	140.0	121.7	125.5	614
6	141325	144.5	123.3	132	631
7	145205	148.5	124.8	136.5	639
8	148513	151.9	126.1	140	648
9	151493	154.9	127.2	145	659
10	154062	157.5	128.2	151.5	673
11	156397	159.9	129.0	153	679
12	158499	162.1	129.5	155.5	684
13	160485	164.1	130.3	158.5	688
14	162317	166.0	130.8	160	691
15	163998	167.7	131.3	163	690
16	165547	169.3	131.7	165	693
17	167088	170.8	132.2	167	702
18	168516	172.3	132.5	168.5	706
19	169817	173.6	132.7	170	709
20	171127	175.0	133.0	172	704

cobertura alta para los grafos de referencia y una cobertura baja o nula para los grafos aleatorios.

5.2.1 CONFIGURACIÓN EXPERIMENTAL

Para medir la cobertura, utilizamos la métrica de *exhaustividad*, ya que esta métrica representa, mediante un valor acotado de 0-1, la completitud de un conjunto A con respecto a una clase de referencia B . Es decir, se evalúa que todos los elementos de B estén también en A . Esta métrica es común en tareas de clasificación

y recuperación de información.

Para calcular exhaustividad, se obtiene el cociente entre los verdaderos positivos y la cantidad de elementos presentes en la clase de referencia (es decir, entre la suma de los verdaderos positivos y los falsos negativos). En nuestro caso, la clase de referencia estaría dada por un grafo de prueba. Por lo tanto, la exhaustividad entre un grafo de dependencia \mathcal{D}_h con nivel h y un grafo de prueba \mathcal{P}_i se calcula como

$$\begin{aligned} \text{Exhaustividad}(\mathcal{D}_h, \mathcal{P}_i) &= \frac{v_p}{v_p + f_n} \\ &= \frac{|E(\mathcal{D}_h) \cap E(\mathcal{P}_i)|}{|E(\mathcal{P}_i)|}, \end{aligned} \tag{5.1}$$

donde v_p representa la cantidad de verdaderos positivos, f_n representa la cantidad de falsos negativos y $E(\mathcal{G})$ representa el conjunto de conexiones de un grafo \mathcal{G} . El valor de la exhaustividad oscila entre 0 y 1, obteniéndose el valor mínimo cuando $E(\mathcal{D}_h)$ y $E(\mathcal{P}_i)$ son completamente diferentes y obteniéndose el valor máximo cuando, por el contrario, son iguales.

En este caso, solamente se evalúa la presencia de las conexiones de los grafos de dependencia en los grafos de prueba. Es decir, no se están considerando los pesos de estas conexiones. Una razón para esto es que la relevancia de la fuerza se asocia más con el estudio de la dependencia y el propósito de esta investigación se enfoca más en el hallazgo de la presencia de una relación entre microservicios.

Tomando en cuenta lo anterior, se calculó la exhaustividad entre cada uno de los grafos de dependencia generados ($\mathcal{D}_1 \dots \mathcal{D}_{20}$) y cada uno de los grafos de prueba, incluyendo los de referencia ($R_1 \dots R_{23}$) y los aleatorios ($A_1^0 \dots A_{23}^{50}$). A continuación, se describe la metodología utilizada para obtener estos grafos de prueba. Como veremos, se obtuvo un conjunto de 23 grafos de referencia, y por cada uno de estos grafos, se generaron 50 grafos aleatorios.

5.2.1.1 GENERACIÓN DE GRAFOS DE REFERENCIA

Cada uno de los grafos de referencia fue generado a partir de flujos de negocio conocidos y representativos de las operaciones de la compañía. Estos flujos de negocio son determinados por los equipos encargados de probar la funcionalidad (*testers*) en las pruebas de aceptación de usuario, por lo que la relevancia de los mismos es alta. Los flujos se encuentran almacenados en una herramienta de administración de pruebas, y el nivel de granularidad es a nivel de paso, lo cual asegura que cada ejecución sea similar y los resultados también. Por otra parte, dentro de la compañía está implementada la práctica de la automatización de pruebas funcionales con la meta de reducir el tiempo de pruebas y evitar el error humano en la ejecución. Esta última característica hace estos flujos idóneos para la experimentación.

Se buscó que los flujos de negocio seleccionados tuvieran casos de prueba automatizados a través de herramientas especializadas que usa la compañía, tales como *Katalon Studio* (<https://www.katalon.com/>). Para obtener estos flujos, se hizo una ejecución aislada de pruebas en secuencia. Típicamente se utiliza una infraestructura basada en *contenedores Dockers* para la ejecución masiva de pruebas automatizadas; sin embargo, para el propósito de experimentación, la ejecución se hizo de caso de prueba por caso de prueba de manera local. Se monitoreó la ejecución en pantalla en busca de errores de ejecución y, si existía algún error, la prueba se reiniciaba hasta obtener un resultado libre de errores.

Se utilizó un analizador de red para capturar el tráfico del cliente con el servidor, como muestra la Figura 5.7. Durante la ejecución local, se utilizó *fiddler* (<https://www.telerik.com/fiddler>) para analizar y registrar el tráfico de la computadora en ejecución y el servidor. Se almacenaron los registros de ejecución de la herramienta de automatización y los registros generados del tráfico al servidor.

Se utilizó un script en Python para filtrar solamente las llamadas a los **APIs** (**HTTP + API + Operación**) en cada flujo registrado (código mostrado en 5.8). Este paso es crucial para separar la información irrelevante de la que es clave; para

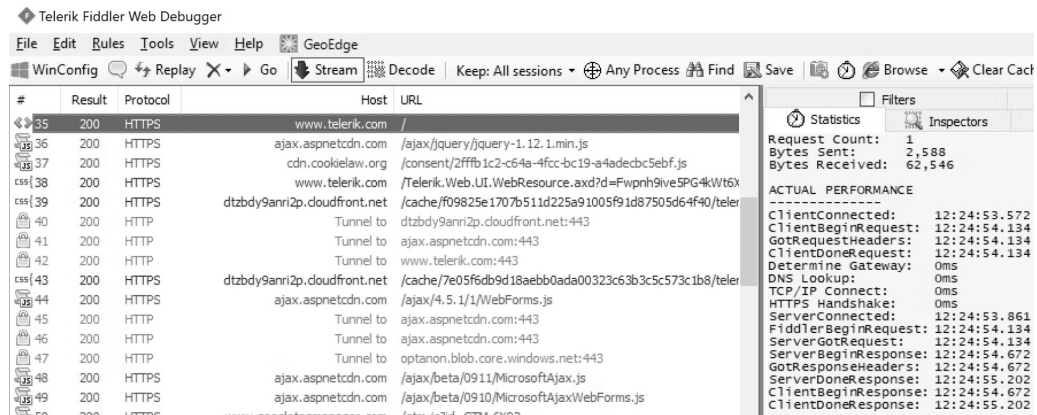


Figura 5.7: Herramienta de monitoreo de redes

esto se hizo una discriminación por parte del servidor donde se hizo la llamada: <https://qa.cemexgo.com/api>.

```
import os

def getListOfFiles(dirName):
    listOfFile = os.listdir(dirName)
    allFiles = list()
    for entry in listOfFile:
        fullPath = os.path.join(dirName, entry)
        if os.path.isdir(fullPath):
            allFiles = allFiles + getListOfFiles(fullPath)
        else:
            allFiles.append(fullPath)
    return allFiles

dirName = 'QA_KT'
files=[]
i=0
listOfFiles = getListOfFiles(dirName)

for file in listOfFiles:
    print(file)
    with open(file, encoding="utf8") as f:
        content = f.readlines()
        for i in content:
            if "https://qa.cemexgo.com/api" in i:
                print(file, " ", i)
```

Figura 5.8: Código de python para procesamiento

Se cargaron todos los resultados en una tabla y se utilizó el catálogo de acti-

vidades (A) generado anteriormente para cotejar la existencia de operaciones (ver Figura 5.3). De este modo, se obtuvieron 23 vectores de flujo de negocio. Cada flujo contiene secuencias de operaciones (actividades) válidas que—teóricamente—deberían de encontrarse en \mathcal{L} . La longitud y cantidad de actividades únicas de cada flujo se puede apreciar en la Tabla 5.3. La generación de grafos y trazas aleatorias a partir de estos flujos se describe a continuación.

Tabla 5.3: Características de los flujos de negocio obtenidos

Flujo	Longitud	Actividades	Flujo	Longitud	Actividades
1	199	33	12	29	17
2	39	19	13	26	13
3	155	32	14	25	16
4	19	13	15	26	14
5	41	14	16	26	15
6	19	12	17	26	14
7	101	13	18	36	16
8	14	12	19	23	15
9	163	22	20	24	15
10	126	57	21	23	13
11	29	17	22	23	13
			23	23	15

La cantidad de eventos registrados en las 23 secuencias varían de 14 a 199. Cada secuencia puede tener una longitud (cantidad de eventos) distinta a la cantidad de actividades (únicas). Por ejemplo, la secuencia #3 tiene una longitud de 155 eventos, de los cuales sólo 32 son únicos.

Para extraer grafos de referencia a partir de los flujos de negocio, se generaron conexiones por cada par de invocaciones consecutivas, de tal manera que $e = (a_i, a_{i+1})$, donde $a_i, a_{i+1} \in A$, i.e. dos invocaciones a microservicios que van seguidas. Por ejemplo, para un flujo $\langle 2, 3, 5, 8 \rangle$, tendríamos las conexiones $(2, 3)$, $(3, 5)$ y $(5, 8)$. Se generó un grafo por flujo.

5.2.1.2 GENERACIÓN DE GRAFOS ALEATORIOS

Para la generación de grafos aleatorios, se escogió el modelo Erdős-Rényi, ya que es uno de los más conocidos y se consideró que sería una opción adecuada para modelar grafos de referencia con relaciones espurias. En este caso, se consideró la variante de este modelo que maneja una cantidad fija de conexiones. En ese sentido, se utilizaron los parámetros (n, m) para la generación, donde n es la cantidad de nodos y m es la cantidad de conexiones; cada una de estas conexiones se generó entre un par de nodos con una distribución uniforme. Por ejemplo, para los parámetros $(20, 100)$, se tendría una generación de 100 conexiones aleatorias en un grafo con 20 nodos.

Para implementar este modelo, nos basamos en secuencias aleatorias de los mismos tamaños que nuestros 23 flujos de negocio. Por lo tanto, para cada secuencia de flujo de negocio con longitud L , se crearon 50 secuencias aleatorias de esta misma longitud. De este modo, para una secuencia en particular—por ejemplo, la #3 (ver la Tabla 5.3)—que tiene una longitud de 155 eventos, se crearon 50 secuencias también de 155 eventos cada una, tomando para ello elementos del conjunto A de forma aleatoria. Una vez teniendo estas secuencias, se extrajeron los grafos aleatorios de la misma manera en que se extrajeron los grafos de referencia: tomando pares de eventos consecutivos y formando conexiones a partir de dichos pares. De esta manera, se generaron $23 \times 50 = 1,150$ grafos aleatorios.

La cantidad de grafos aleatorios por cada grafo de referencia se escogió para dar una mayor estabilidad a los resultados, ya que se calcularía la exhaustividad promedio de cada uno de los 23 conjuntos de grafos aleatorios. Esto, a su vez, ayuda a minimizar resultados atípicos que puedan dar pie a una interpretación errónea del experimento.

5.2.2 RESULTADOS Y DISCUSIÓN

Los resultados iniciales de la validación de exhaustividad para las secuencias de prueba versus su contraparte de las secuencias aleatorias muestran que hay una mejora significativa. Una intuición sobre esto podría desarrollarse con los resultados de la Figura 5.9. Esta figura muestra la comparación en términos de resultados de exhaustividad por *conjuntos de prueba* y el valor promedio obtenido por los 50 *conjuntos aleatorios* en contraparte a cada flujo de prueba teniendo un valor $h = 1$, es decir, con un salto entre eventos.

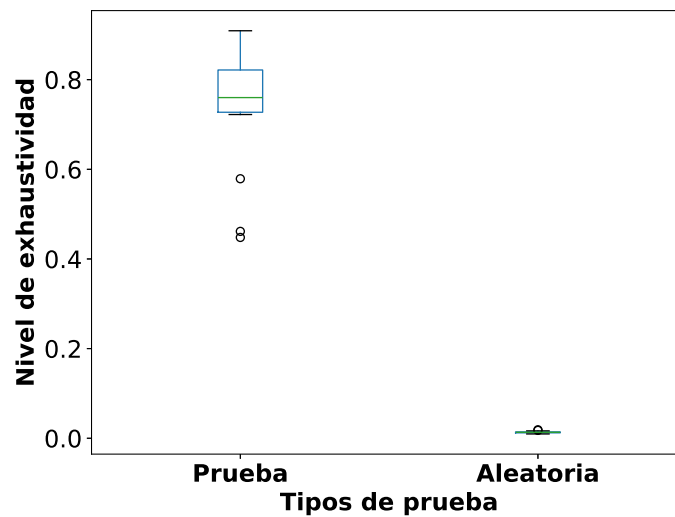


Figura 5.9: Resultados de exhaustividad desde $h = 1$ hasta $h = 20$)

En la Figura 5.9, es posible observar que los valores de exhaustividad obtenidos con los grafos de referencia son mayores que los obtenidos con los grafos aleatorios.

En la Tabla 5.4 se hace un resumen de los valores de exhaustividad obtenidos con los grafos de prueba por cada nivel de h . En la Figura 5.10, se puede observar los resultados de exhaustividad para las secuencias de prueba para cada valor de h , y se puede apreciar que para $h > 1$ el valor de exhaustividad presenta una mejora.

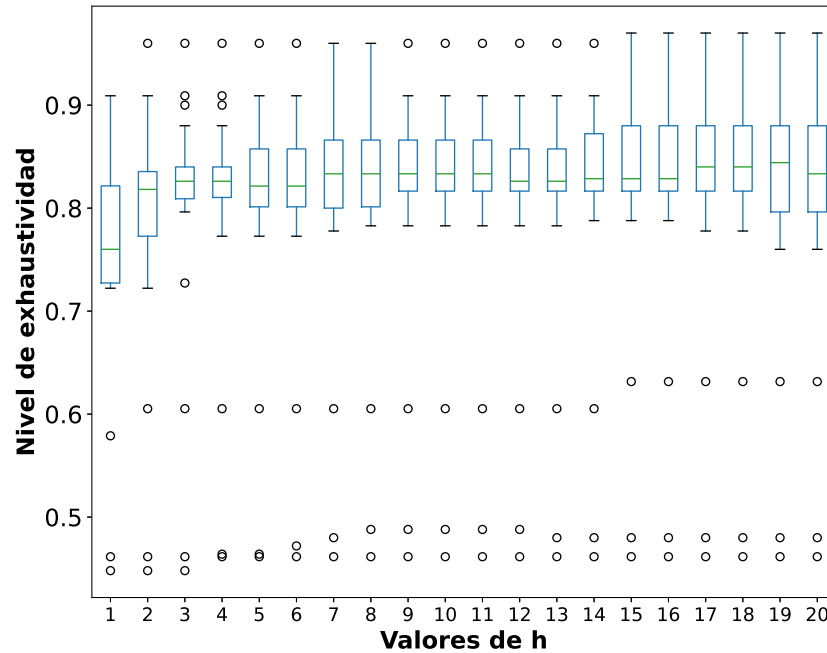


Figura 5.10: Resultados de exhaustividad con diferentes niveles de h

Para saber si esta diferencia es estadísticamente significativa, se ejecutaron varias pruebas y se obtuvieron los siguientes resultados para el escenario donde $h = 1$:

- En los estadísticos descriptivos de la muestra de prueba y los datos aleatorios, se puede observar una diferencia en las medias de ambas poblaciones (0.752 y 0.013 respectivamente).
- En pruebas de normalidad los resultados se pueden verificar en anexo que se puede consultar vía el código QR de la figura A.1.
 - Tanto la prueba *Kolmogorov-Smirnov* como la *Shapiro-Wilk* muestran valores de significación menores a 0.05 (0.0013 y 0.00252 respectivamente), por lo que los resultados de exhaustividad de la muestra de prueba no presenta una distribución normal.
 - En las gráficas Q-Q Normal y Q-Q normal sin tendencia se puede observar esto al igual (anexo que se puede consultar vía el código QR de la figura

Tabla 5.4: Resumen de los resultados de exhaustividad

Resultados de exhaustividad por nivel de salto de evento (h)								
Datos de muestra frente a datos aleatorios					h versus $h = 1$			
h	Promedio		Pruebas		Wilcox Rango Positivo	Two-sided Significancia asintótica	Wilcox Rangos (-,+,=)	Dos colas Significancia asintótica
	datos Prueba	Aleatorios	Kolmogorov Smirnova	Shapiro Wilk				
2	0.784	0.015	0.00075	0.00051	23	0.00003	(2, 12, 9)	0.00284
3	0.796	0.016	0.00000	0.00004	23	0.00003	(1, 14, 8)	0.00146
4	0.798	0.015	0.00000	0.00002	23	0.00003	(1, 16, 6)	0.00071
5	0.798	0.016	0.00000	0.00002	23	0.00003	(0, 16, 7)	0.00043
6	0.799	0.017	0.00000	0.00004	23	0.00003	(0, 16, 7)	0.00043
7	0.802	0.016	0.00002	0.00005	23	0.00003	(0, 16, 7)	0.00043
8	0.803	0.017	0.00000	0.00005	23	0.00003	(0, 16, 7)	0.00043
9	0.805	0.016	0.00000	0.00003	23	0.00003	(0, 16, 7)	0.00029
10	0.805	0.017	0.00000	0.00003	23	0.00003	(0, 17, 6)	0.00029
11	0.805	0.018	0.00000	0.00003	23	0.00003	(0, 17, 6)	0.00029
12	0.802	0.017	0.00000	0.00003	23	0.00003	(0, 17, 6)	0.00029
13	0.801	0.017	0.00000	0.00002	23	0.00003	(0, 17, 6)	0.00029
14	0.804	0.017	0.00000	0.00003	23	0.00003	(0, 17, 6)	0.00029
15	0.812	0.018	0.00002	0.00003	23	0.00006	(0, 17, 6)	0.00029
16	0.812	0.019	0.00002	0.00006	23	0.00006	(0, 17, 6)	0.00029
17	0.813	0.017	0.00003	0.00006	23	0.00006	(0, 16, 7)	0.00043
18	0.813	0.017	0.00003	0.00003	23	0.00006	(0, 17, 6)	0.00029
19	0.811	0.020	0.00053	0.00015	23	0.00003	(2, 15, 6)	0.00071
20	0.807	0.017	0.00024	0.00022	23	0.00003	(2, 15, 6)	0.00071

A.1).

- En la Prueba de rangos con signo de Wilcoxon (tabla 5.4) se observó que los 23 flujos de muestra presentaron positivo a la afirmación *cobertura con grafos de referencia* $>$ *cobertura con grafos aleatorios*.
- Encontramos que la significancia asintótica bilateral (0.00003) es menor al coeficiente de confianza 0.05 , usando la prueba de rangos con signo de Wilcoxon, por lo que podemos deducir que hay una diferencia estadísticamente significativa entre los resultados de cobertura de los grafos de dependencia con los grafos de referencia y los resultados de cobertura de los grafos de dependencia con los grafos aleatorios.

Con base en el párrafo anterior, podemos decir que los resultados de validación sobre el grafo de dependencia relajado son estadísticamente diferentes del azar.

5.2.2.1 RESULTADOS CON DIFERENTES NIVELES

Para efectos de prueba, se incrementó el valor de h desde el valor de 2 hasta el valor de 20. Para cada valor se repitieron las mismas pruebas que para $h = 1$, con la agregación de una prueba no paramétrica de comparación de medias de muestras relacionadas (*Prueba de rangos con signo de Wilcoxon*) para comparar todos los valores de $h > 1$.

En la Tabla 5.4, podemos observar en las columnas comprendidas bajo el apartado de *Datos de muestra frente a datos aleatorios* que cualquier nivel de h tiene diferencia estadística respecto a los datos aleatorios, el rango positivo de *Wilcox* lo muestra soportado por la prueba de *dos colas de significancia asintótica* cuyo valor es inferior al margen de error de 0.05 . Por otro lado, cualquier valor de h superior a 1 muestra mejoría en términos estadísticos sobre el número de relaciones encontradas, especialmente los valores de h entre 10 y 16 (mostrado en los datos de la Tabla 5.4).

Con la intención de hacer más digerible la información por cada valor incrementado, se muestra la Figura 5.10 comparando el resultado de pruebas de exhaustividad de los datos de prueba y aleatorios para el valor de h bajo prueba y una gráfica más mostrando visualmente los resultados de exhaustividad de los datos de prueba de $h = 1$ contra los resultados de los datos sobre el h bajo prueba, esto para ganar una intuición visual de los mismos. Cada resultado está referenciado a la tabla de resultados mostrados en el anexo que se puede consultar vía el código QR de la Figura A.1.

Los resultados obtenidos nos permiten afirmar existe una diferencia estadísticamente significativa al incrementar el nivel relajación h respecto a $h = 1$.

5.3 PRUEBAS DE DESCUBRIMIENTO

El objetivo de estas pruebas fue demostrar que las conexiones descubiertas—es decir, aquellas que están en el grafo de dependencia, pero no están en los grafos de referencia—no están presentes en el grafo por azar (i.e., que no son ruido o relaciones espurias), sino que son producto de una minería sobre la bitácora. En ese sentido, nuestro propósito fue mostrar que estas relaciones no están en la planeación original, pero existen dentro de la operación diaria de la compañía. Para evaluar la autenticidad de estas conexiones, las comparamos contra diferentes modelos de configuración (explicados en la Sección 2.2.1). Por lo tanto, asumimos que una diferencia estadísticamente significativa entre las conexiones descubiertas y las conexiones en un modelo de configuración es indicativa de dicha autenticidad.

5.3.1 CONFIGURACIÓN EXPERIMENTAL

Si tenemos un grafo del modelo de configuración (\mathcal{M}) en forma de matriz de adyacencias donde u representa un nodo origen y v un nodo destino, entonces $|u >_{\mathcal{L}} v|$ representa el conteo de una relación entre los microservicios de estos nodos. Por tanto, \mathcal{M}_h representa una matriz donde tenemos el conteo sobre \mathcal{L} (bitácora de eventos) con un nivel h y es la que será evaluada en esta prueba. Para simplificar el análisis de datos sobre las matrices de adyacencia, tanto del grafo de dependencia relajada como para el grafo de modelo de configuración para distintos valores de h , se aplicó una reducción de datos a través de la sumatoria de cada renglón de cada matriz de adyacencia. Por cada grafo de dependencia relajado para determinado nivel h , se crearon 50 grafos con el modelo de configuración. Esos 50 grafos en forma de matriz de adyacencia se sumaron y promediaron para tener una muestra única para cada nivel h contra la cual comparar el grafo de dependencia relajado.

Se ejecutaron una serie de pruebas para validar las características de los datos

y avalar los resultados. El primer conjunto de pruebas fue destinado a determinar la normalidad de cada conjunto de datos, lo cual ayudó a determinar qué método de diferenciación estadística es el más adecuado. Los métodos propuestos para determinar la normalidad son:

- Prueba de normalidad de Kolmogorov-Smirnov
- Prueba de Shapiro-Wilk

Una vez que se determinó si los resultados representan una distribución normal o no, se aplicaron distintas pruebas sobre diferencias de población:

- En caso de normalidad
 - Prueba t -Student
- En caso de no normalidad
 - Pruebas no paramétricas para muestras relacionadas

En la siguiente sección, se muestran los resultados numéricos de las siguientes pruebas:

- Las medias obtenidas para cada muestra.
- La significación bilateral para determinar la normalidad usando las pruebas de Kolmogorov-Smirnova y Shapiro-Wilk.
- Los resultados de las prueba de rangos con signo de Wilcoxon.
- Significancia asintótica bilateral entre las muestras.

Se ejecutaron estas pruebas para los niveles $h = 1$ hasta $h = 5$, ya que se observó un comportamiento lineal en los resultados de medias (Figura 5.11). Las pruebas de normalidad y significancia entre muestras no cambiaron.

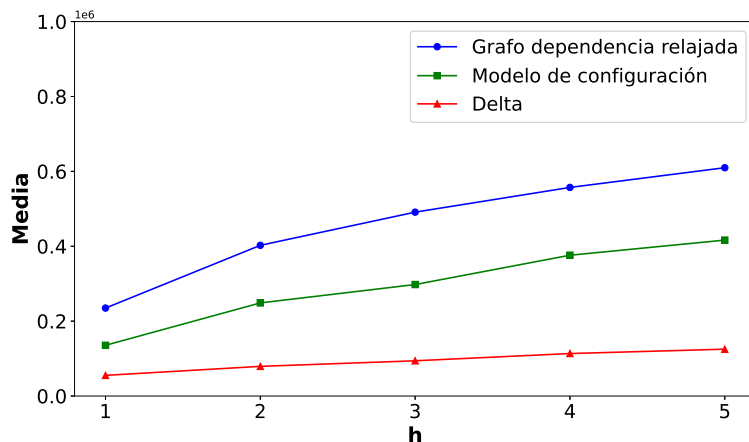


Figura 5.11: Medias de muestras con niveles de $h = 1$ hasta $h = 5$

En base a los resultados mostrados, podemos decir que hay diferencias estadísticamente significativas entre las relaciones encontradas por el grafo de dependencia relajado para cada nivel probado contra las relaciones encontradas al azar. Los detalles y gráficas generadas para el resumen siguiente se puede consultar vía el código QR de la Figura A.1.

5.3.2 RESULTADOS Y DISCUSIÓN

En la Tabla 5.5, se pueden encontrar los resultados de las pruebas estadísticas para los cinco niveles de h mencionados anteriormente. En esta tabla se puede observar que el promedio de relaciones de eventos encontradas en ambas matrices siempre es mayor para la matriz de secuencia relajada que para el modelo de configuración, incluyendo las relaciones conocidas compartidas. Esto podría interpretarse como que las relaciones adicionales encontradas por casualidad no contribuyen significativamente al recuento de relaciones en el registro de eventos \mathcal{L} .

Además, los resultados de las pruebas de Kolmogorov-Smirnova y Shapiro-Wilk que se muestran en la misma tabla respaldan la diferencia estadística entre ambas poblaciones, teniendo rangos positivos en la prueba de *Wilcox*, entre el 95.5 % al

Matriz de secuencia relajada versus modelo de configuración						
h	Promedio		Pruebas		Rangos Wilcoxon (-,+,=)	Significancia Asintótica de dos colas
	Matriz relajada	Modelo Configuración	Kolmogorov Smirnova	Shapiro Wilk		
1	269316.72	160375.53	0.000	0.000	(6, 945, 27)	0.000
2	402471.08	264397.94	0.000	0.000	(13, 952, 13)	0.000
3	491033.18	313501.38	0.000	0.000	(25, 942, 11)	0.000
4	557084.25	357526.11	0.000	0.000	(30, 937, 11)	0.000
5	609708.03	403090.79	0.000	0.000	(33, 934, 11)	0.000

Tabla 5.5: Resumen de los resultados de la validación

97.3% de las operaciones disponibles encontradas en las matrices de prueba.

En la Figura 5.12 podemos observar que el número de conteos es mayor en las relaciones encontradas en grafo de dependencia relajada \mathcal{D}_h que en las encontradas en el modelo de configuración para el mismo nivel de relajación h . Adicionalmente, se calculó una tercera serie mostrada en la gráfica como *serie C* a partir de las relaciones encontradas en el modelo de configuración pero que no estaban en el grafo de dependencia relajada. Como se puede observar, este valor que representa el conteo donde no hay relaciones comunes entre ambas matrices es más bajo en todos los casos que el valor encontrado en el modelo de configuración, reforzando el punto de que las operaciones comunes aumentan el conteo para la matriz generada bajo el modelo de configuración.

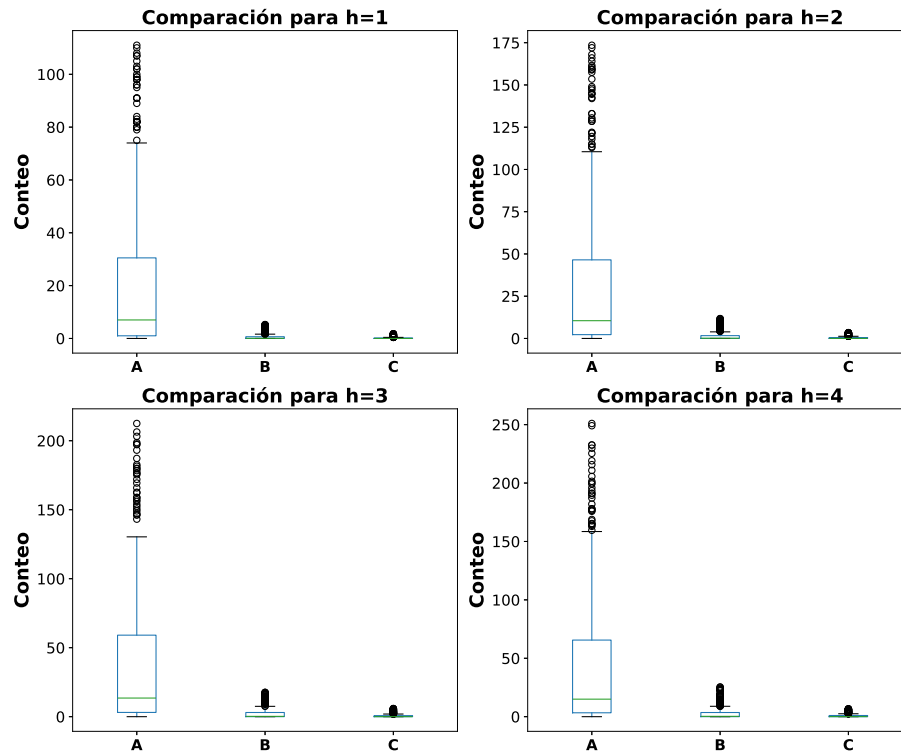


Figura 5.12: A - Grafo de dependencia relajada, B - Grafo de modelo de configuración, C - Relaciones en el modelo de configuración no presentes en el grafo de dependencia relajada

Basándonos en los resultados mostrados, podemos decir que existe una diferencia estadísticamente significativa entre las relaciones encontradas por el grafo de dependencia relajado y las relaciones en los grafos aleatorios. Asimismo, se reporta una mejora en el número de relaciones cuando el valor de h se incrementa.

5.4 RESUMEN

En este capítulo se revisaron dos experimentos destinados a determinar si las relaciones encontradas por el método propuesto y sus mejoras podrían distinguirse del azar. Se siguieron dos caminos distintos, el primero basado en pruebas de exhaustividad en el cual se utilizaron datos obtenidos de muestras que nos ayudaron

a tener una base sobre la cual se obtuvo una referencia para medir las relaciones encontradas.

Los resultados de este primer experimento demostraron que las relaciones encontradas se distinguen estadísticamente del azar, y representan una forma de inferir nuevas relaciones entre microservicios. Adicionalmente, los resultados mostraron que la mejora propuesta denominada nivel de relajamiento (h) proporcionó una mejora estadísticamente comprobable respecto a un nivel 1 o sin relajamiento.

El segundo método propuesto se basó en modelos de configuración. Nos proporcionó una forma de valorar las relaciones descubiertas sin la utilización de datos externos como fue el primer método. El modelo de configuración nos permitió valorar si las relaciones descubiertas en grafos con las mismas características base del grafo de dependencia tales como: número de aristas, nodos y el mismo grado por nodo, generados de manera aleatoria eran distinguibles de las relaciones encontradas en el grafo de dependencia relajada sin y con relajamiento.

Los resultados mostrados en este capítulo respaldan que en ambos métodos, los resultados obtenidos por medio del grado de dependencia relajados son distinguibles del azar. En el siguiente capítulo, se hará un énfasis en las contribuciones que los resultados obtenidos respaldan.

CAPÍTULO 6

CONCLUSIONES Y TRABAJO FUTURO

6.1 RESUMEN

Las arquitecturas de microservicios representan una evolución que resolvió limitaciones clave con respecto a modelos anteriores, como los modelos monolíticos, en términos de escalabilidad, flexibilidad y velocidad para construir e implementar aplicaciones. No obstante, también representan algunos desafíos importantes derivados de sus atributos clave, como la independencia y la composición a nivel de proceso comercial.

Un desafío al que nos enfrentamos durante la elaboración de esta investigación fue el cómo determinar las relaciones (conexiones) en una red de microservicios usando una bitácora restringida, es decir, teniendo sólo la llamada o evento y una firma de tiempo de cuándo se registró en el sistema.

Una forma intuitiva de entender estas relaciones es a través de una estructura de datos como los grafos, que nos permiten definir múltiples relaciones, dirección de la relación e incluso su fuerza. Al trasladar el problema de entender las relaciones de microservicios al área de teoría de grafos y redes complejas, se abre la posibilidad a técnicas como análisis de centralidad, comunidades, grados y algoritmos bien establecidos en este marco de trabajo.

Para la transformación de una bitácora simple con los atributos antes mencionados (evento y firma de tiempo), se buscaron marcos de trabajo que nos dieran alguna forma de entender y medir la solución a este problema.

La minería de procesos ofrece una base de conocimiento para explotar los datos generados como consecuencia de su uso natural. Sin embargo, sólo en algunos casos, toda la información requerida para el descubrimiento de procesos está disponible en la forma ideal, tal como el identificador de traza, evento y firma de tiempo. Una forma experimental de analizar y aprovechar la información implícita en los registros de eventos es la minería de procesos, que, en muchos casos, como el de este estudio, sirve como punto de partida.

En el transcurso de esta investigación, se trataron diferentes marcos teóricos que nos ayudaron a la creación de un grafo de dependencia relajado. Asimismo, se definió una metodología para diseñar y evaluar experimentos.

6.2 RESPUESTA A LAS PREGUNTAS DE INVESTIGACIÓN

Las preguntas de investigación planteadas fueron las siguientes:

- *¿Cómo se puede procesar una bitácora de microservicios, de tal manera que pueda extraerse un grafo que represente las relaciones entre microservicios?*
 - Se simplificó la llamada de cada microservicio a la forma mínima que pudiera identificarse y repetirse.
 - Se generó un catálogo único de operaciones que permitiera conocer el universo de eventos esperados.
 - Se convirtió la bitácora original en una secuencia numérica que fue procesada a través de matrices y arreglos.
 - El proceso descrito es repetible en casos similares, por lo que podemos

responder que sí se diseñó e implementó un mecanismo para procesar una bitácora masiva de registros (mas de 400 millones).

- *¿Cómo se puede inferir la relación entre dos microservicios utilizando una bitácora simple?*
 - Durante esta investigación se documentó y generó evidencia que con el uso de un grafo de dependencia es posible resolver esta cuestión, y adicionalmente se demostró que usando un modelo de relajamiento en la construcción de este grafo mejoran los resultados.
- *¿Cómo se puede medir la calidad los resultados?*
 - Para esta investigación se diseñaron e implementaron dos métodos:
 1. Muestras que eran conocidas como verdaderas contra muestras aleatorias basadas en operaciones validas (prueba de exhaustividad).
 2. Grafos aleatorios que validaban los resultados encontrados (modelo de configuración).
 - Los resultados documentados nos ayudan a responder afirmativamente esta pregunta.

6.3 CONTRIBUCIONES

En resumen, nuestras aportaciones fueron las siguientes:

Enfoque sin identificador de traza. El minero heurístico considera los tres componentes de descubrimiento del proceso: identificador de traza, evento y firma de tiempo. El método propuesto se utiliza unicamente para evento y firma de tiempo. El estado del arte para casos similares requiere estructuras adicionales a la bitácora.

Enfoque de meta-traza. Se trata todo el registro de eventos entero como un solo proceso, ya que no hay identificadores de traza. Este enfoque está respaldado por datos masivos y patrones de repetición esperados.

Enfoque relajado. Implementando una contribución degradada para el grafo de secuencia basado en la distancia del evento en los registros y el conteo de frecuencias. La contribución degradada es producto de un nivel de relajación h incluido en las medidas de secuencia y dependencia.

6.4 POSIBLES APLICACIONES

Como se mencionó en el planteamiento del problema de esta investigación, el objetivo de la misma era obtener una plataforma que sirviera como entrada para generar análisis desde la perspectiva de la teoría de redes complejas, ya que en la práctica se ha demostrado que las redes de microservicios se comportan como tal.

Desde ese punto, una vez que hemos determinado que es posible crear un grafo dirigido con pesos al que llamamos grafo de dependencia relajado, este se puede utilizar como entrada entre otros usos, para los siguientes:

- Análisis de riesgos basados en la centralidad de APIs o de microservicios críticos para el negocio
- Análisis de duplicidad de funcionalidad implementada
- Análisis y diseño de flujos de prueba basados en servicios (pruebas integrales o de punto a punto)
- Análisis de dependencias entre componentes
- Análisis de impacto sobre cambios y depreciaciones de funcionalidad
- Planeación de capacidad de arquitectura basada en el tráfico de las aristas de la red

- Análisis de simplificación funcional

6.5 TRABAJO FUTURO

Hay muchas variables que pueden explorarse aún en este campo, así como interrogantes que no se abordaron plenamente durante el desarrollo de esta investigación. Algunos de esos temas donde es posible realizar investigaciones futuras relacionadas son:

- Determinar el nivel mínimo de datos para obtener resultados semejantes.
- Desarrollar métodos alternos para lidiar con la concurrencia en bitácoras de este tipo.
- Investigar alternativas para manejar bitácoras sin agrupador de caso.
- Desarrollar métodos para medir y reducir el ruido en la bitácora que generan dependencias espurias.
- Determinar qué otros marcos de trabajo diferentes a minería de procesos pueden llegar a resultados semejantes.
- Desarrollar un modelo que permita el análisis en tiempo real de dependencias usando bitácoras.
- Investigar cómo se puede complementar esta investigación con temas de monitoreo de infraestructura.
- Uso de inteligencia artificial en el descubrimiento de relaciones.

6.6 COMENTARIOS FINALES

Los modelos de negocio actuales en entornos globalizados y distribuidos plantean retos constantes a los departamentos de tecnología de las organizaciones. Cada vez es más común ver el concepto de deuda técnica crecer y tener que decidir si entre innovar para mantenerse vigentes o buscar estabilidad técnica que le permita a la organización seguir operando.

En este contexto, han nacido múltiples herramientas y servicios comerciales que ofrecen a las compañías un punto de equilibrio entre ambas necesidades: crecer y ofrecer de manera continua un servicio determinado.

Esta investigación nace precisamente bajo esa premisa, bajo la pregunta: *¿Cómo podemos determinar el nivel de riesgo que tenemos si nos falla un determinado servicio?* El viaje y esfuerzo que llevó poderla resolver y presentar algo tangible y verificable ha sido significativo. Sin embargo, la investigación desarrollada y los resultados presentados, sólo son una parte de la respuesta a la pregunta antes planteada.

El mundo de las tecnologías de información ha estado y estará en constante evolución. Nuevos descubrimientos hacen obsoletos a previos y abren posibilidades a soluciones novedosas y efectivas. *¿Como se podrá implementar una solución al problema planteado usando alguna otra tecnología?*—esa pregunta siempre está vigente, y la respuesta, en constante cambio.

APÉNDICE A

ANEXOS

Para efectos de simplificación en términos de longitud, se pone a disposición una amplia sección de detalles de resultados que pueden ser consultados en el siguiente código QR.

También disponible en la siguiente liga <https://drive.google.com/drive/folders/1Rr2PElggVQ1RXQTYKxi1zbyuMmiYBboZ>.

versión 19 Septiembre 2024



Figura A.1: Anexo con detalles de los resultados

BIBLIOGRAFÍA

- ALSHUQAYRAN, N., N. ALI y R. EVANS (2016), «A Systematic Mapping Study in Microservice Architecture», en *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, págs. 44–51.
- ASSUNÇÃO, W. K., J. KRÜGER, S. MOSSER y S. SELAOUI (2023), «How do microservices evolve? An empirical analysis of changes in open-source microservice repositories», *Journal of Systems and Software*, **204**, pág. 111788, URL <https://www.sciencedirect.com/science/article/pii/S0164121223001838>.
- BAYOMIE, D., C. DI CICCIO, M. LA ROSA y J. MENDLING (2019), «A Probabilistic Approach to Event-Case Correlation for Process Mining», en A. H. F. Laender, B. Pernici, E.-P. Lim y J. P. M. de Oliveira (editores), *Conceptual Modeling*, Springer International Publishing, Cham, págs. 136–152.
- BAYOMIE, D., I. HELAL, A. AWAD, E. EZAT y A. ELBASTAWISSI (2016), «Deducing case IDs for unlabeled event logs», en *International Conference on Business Process Management*, Springer, págs. 242–254, URL https://doi.org/10.1007/978-3-319-42887-1_20.
- BEN LAMINE, S. B. A., H. BAAZAOU ZGHAL, M. MARISSA y C. GHEDIRA GUEGAN (2017), «An ontology-based approach for personalized RESTful Web service discovery», *Procedia Computer Science*, **112**, págs. 2127–2136.
- BERANT, J., I. DAGAN y J. GOLDBERGER (2012), «Learning Entailment Relations

- by Global Graph Structure Optimization», *Comput. Linguist.*, **38**(1), págs. 73—111, URL https://doi.org/10.1162/COLI_a_00085.
- BERTON, L., J. VALVERDE-REBAZA y A. DE ANDRADE LOPES (2015), «Link prediction in graph construction for supervised and semi-supervised learning», en *2015 International Joint Conference on Neural Networks (IJCNN)*, págs. 1–8.
- CHAO, Y. y Y. MENG-TING (2010), «Web service composition using graph model», en *2010 2nd IEEE International Conference on Information Management and Engineering*, págs. 656–660.
- CINQUE, M., R. D. CORTE y A. PECCHIA (2021), «Microservices Monitoring with Event Logs and Black Box Execution Tracing», en *2021 IEEE World Congress on Services (SERVICES)*, págs. 12–12.
- Co., I. (2022), «Process Mining», <https://www.ibm.com/cloud/learn/process-mining>.
- CONWAY, M. E. (1968), «How Do Committees Invent?», *Datamation*, URL <http://www.melconway.com/research/committees.html>.
- DAS SARMA, A., A. JAIN y C. YU (2011), «Dynamic Relationship and Event Discovery», en *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining, WSDM '11*, Association for Computing Machinery, New York, NY, USA, págs. 207—216, URL <https://doi.org/10.1145/1935826.1935867>.
- DI FRANCESCO, P. (2017), «Architecting microservices», *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings*, págs. 224–229.
- DOS SANTOS GARCIA, C., A. MEINCHEIM, E. R. FARIA JUNIOR, M. R. DALLAGASSA, D. M. V. SATO, D. R. CARVALHO, E. A. P. SANTOS y E. E. SCALABRIN (2019), «Process mining techniques and applications – A systematic mapping study», *Expert Systems with Applications*, **133**, págs. 260–295, URL <https://www.sciencedirect.com/science/article/pii/S0957417419303161>.

- FANG, C., J. LIU y Z. LEI (2014), «Parallelized user clicks recognition from massive HTTP data based on dependency graph model», *China Communications*, **11**(12), págs. 13–25.
- FRANCESCO, P. D., I. MALAVOLTA y P. LAGO (2017), «Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption», en *2017 IEEE International Conference on Software Architecture (ICSA)*, págs. 21–30.
- GOUGOUX, J. P. y D. TAMZALIT (2017), «From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture», *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings*, págs. 62–65.
- GUO, X., X. PENG, H. WANG, W. LI, H. JIANG, D. DING, T. XIE y L. SU (2020), «Graph-Based Trace Analysis for Microservice Architecture Understanding and Problem Diagnosis», en *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020*, Association for Computing Machinery, New York, NY, USA, págs. 1387—1397, URL <https://doi.org/10.1145/3368089.3417066>.
- HEJDERUP, J., A. VAN DEURSEN y G. GOUSIOS (2018), «Software ecosystem call graph for dependency management», en *ICSE-NIER '18: Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER '18, Association for Computing Machinery, New York, NY, USA, pág. 101–104, URL <https://doi.org/10.1145/3183399.3183417>.
- IBRAHIM, A., S. BOZHINOSKI y A. PRETSCHNER (2019), «Attack Graph Generation for Microservice Architecture», en *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC '19*, Association for Computing Machinery, New York, NY, USA, págs. 1235—1242, URL <https://doi.org/10.1145/3297280.3297401>.
- IEEE (2022), «Process Mining», <https://www.win.tue.nl/ieeetfpm/downloads/Process>

- IMPERVA (2022), «Web Application Firewalls», <https://www.imperva.com/>, fecha de acceso: 2022-06-04.
- IRACHETA, J. y S. E. GARZA (2020), «Generación de un grafo probabilístico en base a bitácoras de micro servicios RESTful», *Aplicaciones de la Computación*, págs. 229–236.
- JACOB, S., B. LEE y Y. QIAO (2019), «Applying Process Mining to Improve Microservices Cyber Security Situational Awareness», Athlon Institute of Technology Graduate School, URL <https://research.thea.ie/handle/20.500.12065/3465>.
- JAMSHIDI, P., C. PAHL, N. C. MENDONCA, J. LEWIS y S. TILKOV (2018), «Microservices: The journey so far and challenges ahead», *IEEE Software*, **35**(3), págs. 24–35.
- JOSUTTIS, N. M. (2007), *SOA in Practice: The Art of Distributed System Design (Theory in Practice)*, O'Reilly Media.
- KAUR, P., H. SINGHAL, A. SAXENA, N. MITTAL y C. DABAS (2021), «PolyGlot Persistence for Microservices-Based Applications», *Int. J. Inf. Technol. Syst. Approach*, **14**(1), pág. 17–32, URL <https://doi.org/10.4018/IJITSA.2021010102>.
- KIRBY, L. J., E. BOERSTRA, Z. J. ANDERSON y J. RUBIN (2021), «Weighing the Evidence: On Relationship Types in Microservice Extraction», en *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, págs. 358–368.
- KURYAZOV, D., D. JABBOROV y B. KHUJAMURATOV (2020), «Towards Decomposing Monolithic Applications into Microservices», en *2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT)*, págs. 1–4.
- LARRUCEA, X., I. SANTAMARIA, R. COLOMO-PALACIOS y C. EBERT (2018), «Microservices», *IEEE Software*, **35**(3), págs. 96–100.

- LI, B., Z. LI y Y. YANG (2021), «Residual attention graph convolutional network for web services classification», *Neurocomputing*, **440**, págs. 45–57, URL <https://www.sciencedirect.com/science/article/pii/S0925231221001715>.
- LI, H., S. LI, J. SUN, Z. XING, X. PENG, M. LIU y X. ZHAO (2018), «Improving API Caveats Accessibility by Mining API Caveats Knowledge Graph», en *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, págs. 183–193.
- LIU, H., Z. CAO y X. ZHANG (2018), «An Efficient Algorithm of Context-Clustered Microservice Discovery», en *Proceedings of the 2nd International Conference on Computer Science and Application Engineering, CSAE '18*, Association for Computing Machinery, New York, NY, USA, págs. 1–6.
- MA, M., J. XU, Y. WANG, P. CHEN, Z. ZHANG y P. WANG (2020), «AutoMAP: Diagnose Your Microservice-Based Web Applications Automatically», en *Proceedings of The Web Conference 2020, WWW '20*, Association for Computing Machinery, New York, NY, USA, págs. 246—258, URL <https://doi.org/10.1145/3366423.3380111>.
- MA, S.-P., C.-Y. FAN, Y. CHUANG, W.-T. LEE, S.-J. LEE y N.-L. HSUEH (2018), «Using Service Dependency Graph to Analyze and Test Microservices», en *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, tomo 02, págs. 81–86.
- MANKOVSKI, S. (2009), *Service Oriented Architecture*, capítulo Service Oriented Architecture, Springer US, Boston, MA, págs. 2633–2636.
- MAO, C. (2009), «A specification-based testing framework for web service-based software», *2009 IEEE International Conference on Granular Computing, GRC 2009*, págs. 440–441.
- MARIADB (2022), «MariaDB Server: The open source relational database», <https://mariadb.org/>, fecha de acceso: 2022-06-04.

- MENG, Y., D. XU, Z. ZHANG y W. LI (2015), «System Dependency Graph Construction Algorithm Based on Equivalent Substitution», en *2015 Eighth International Conference on Internet Computing for Science and Engineering (ICICSE)*, págs. 106–110.
- MICROSOFT (2022a), «AZURE. INVENT WITH PURPOSE», <https://azure.microsoft.com/en-us/>, fecha de acceso: 2022-06-04.
- MICROSOFT (2022b), «Kusto Query Language (KQL) overview», <https://docs.microsoft.com/en-us/azure/data-explorer/kusto/query/>, fecha de acceso: 2022-06-04.
- MUNONYE, K. (2021), «Microservices Data Mining for Analytics Feedback and Optimization», *International Journal of Enterprise Information Systems (IJEIS)*, **17**(1), págs. 22–43.
- NAROCK, T., V. YOON y S. MARCH (2014), «A provenance-based approach to semantic web service description and discovery», *Decision Support Systems*, **64**, págs. 90–99, URL <https://www.sciencedirect.com/science/article/pii/S0167923614001286>.
- NEWMAN, M. (2018), *Networks*, Oxford University Press.
- NEWMAN, M. E. (2002), «The structure and function of networks», *Computer Physics Communications*, **147**(1-2), págs. 40–45.
- NEWMAN, M. E. (2008), «The mathematics of networks», *The new palgrave encyclopedia of economics*, **2**(2008), págs. 1–12.
- NGUYEN, C. D., A. MARCHETTO y P. TONELLA (2011), «Challenges in audit testing of web services», *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2011*, págs. 103–106.
- NI, L., W. LUO, N. LU y W. ZHU (2020), «Mining the Local Dependency Itemset in a Products Network», *ACM Trans. Manage. Inf. Syst.*, **11**(1), URL <https://doi.org/10.1145/3384473>.

- OBERHAUSER, R. y S. STIGLER (2018), «Microflows: Leveraging process mining and an automated constraint recommender for microflow modeling», *Lecture Notes in Business Information Processing*, **309**, págs. 25–48.
- OMER, A. M. y A. SCHILL (2009), «Dependency Based Automatic Service Composition Using Directed Graph», en *2009 Fifth International Conference on Next Generation Web Services Practices*, págs. 76–81.
- PAHL, M.-O., F.-X. AUBET y S. LIEBALD (2018), «Graph-based IoT microservice security», en *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, págs. 1–3.
- PAUTASSO, C., O. ZIMMERMANN, M. AMUNDSEN, J. LEWIS y N. JOSUTTIS (2017), «Microservices in Practice, Part 1: Reality Check and Service Design», *IEEE Software*, **34**(1), págs. 91–98.
- PYTHON (2022), «Python programming language», <https://www.python.org/>, fecha de acceso: 2022-06-04.
- RAILÍĆ, N. y M. SAVIĆ (2021), «Architecting Continuous Integration and Continuous Deployment for Microservice Architecture», en *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)*, págs. 1–5.
- RAJESH, M. *et al.* (2021), «Conformance Checking Techniques of Process Mining: A Survey», *Recent Trends in Intensive Computing*, **39**, pág. 335.
- REI, M. y T. BRISCOE (2011), «Unsupervised Entailment Detection between Dependency Graph Fragments», en *Proceedings of BioNLP 2011 Workshop*, BioNLP '11, Association for Computational Linguistics, USA, págs. 10—18.
- SAKAI, M., K. TAKAHASHI y S. KONDOH (2021), «Method of Constructing Petri Net Service Model Using Distributed Trace Data of Microservices», en *2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, págs. 214–217.

- SINGH, K., H. K. SHAKYA y B. BISWAS (2015), «An Efficient Approach to Discovering Frequent Patterns from Data Cube Using Aggregation and Directed Graph», en *Proceedings of the Sixth International Conference on Computer and Communication Technology 2015, ICCCT '15*, Association for Computing Machinery, New York, NY, USA, págs. 31—35, URL <https://doi.org/10.1145/2818567.2818573>.
- STACKOVERFLOW (2022), «A public platform building the definitive collection of coding questions and answers», <https://stackoverflow.com/>, fecha de acceso: 2022-06-04.
- TAIBI, D. y K. SYSTÄ (2019), «From monolithic systems to microservices: A decomposition framework based on process mining», *CLOSER 2019 - Proceedings of the 9th International Conference on Cloud Computing and Services Science*, págs. 153–164.
- VAN DER AALST, W., T. WEIJTERS y L. MARUSTER (2004), «Workflow mining: discovering process models from event logs», *IEEE Transactions on Knowledge and Data Engineering*, **16**(9), págs. 1128–1142.
- WANG, H., C. SHAH, P. SATHAYE, A. NAHATA y S. KATARIYA (2019), «Service Application Knowledge Graph and Dependency System», en *2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*, págs. 134–136.
- WANG, R. y J. ZHANG (2017), «Aspect Oriented Integration Testing Method Based on Module Dependency Graph», en *2017 International Conference on Computer Technology, Electronics and Communication (ICCTEC)*, págs. 976–982.
- WEIJTERS, A., W. AALST y A. MEDEIROS (2006), *Process Mining with the Heuristics Miner-algorithm*, tomo 166, Technische Universiteit Eindhoven.
- WEIJTERS, A. J. y J. T. RIBEIRO (2011), «Flexible heuristics miner (FHM)», en *IEEE SSCI 2011: Symposium Series on Computational Intelligence - CIDM*

- 2011: 2011 IEEE Symposium on Computational Intelligence and Data Mining*, April 2011, págs. 310–317.
- YING, L. (2010), «A Method of Automatic Web Services Composition Based on Directed Graph», en *2010 International Conference on Communications and Mobile Computing*, tomo 1, págs. 527–531.
- ZERBINO, P., A. STEFANINI y D. ALOINI (2021), «Process Science in Action: A Literature Review on Process Mining in Business Management», *Technological Forecasting and Social Change*, **172**, pág. 121021, URL <https://www.sciencedirect.com/science/article/pii/S0040162521004534>.
- ZHU, J., S. HE, J. LIU, P. HE, Q. XIE, Z. ZHENG y M. R. LYU (2019), «Tools and Benchmarks for Automated Log Parsing», en *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, págs. 121–130.
- ZUO, Y., X. ZHU, J. QIN y W. YAO (2021), «Temporal Relations Extraction and Analysis of Log Events for Micro-service Framework», en *2021 40th Chinese Control Conference (CCC)*, págs. 3391–3396.
- ÁLVARO BRANDÓN, M. SOLÉ, A. HUÉLAMO, D. SOLANS, M. S. PÉREZ y V. MUNTÉS-MULERO (2020), «Graph-based root cause analysis for service-oriented and microservice architectures», *Journal of Systems and Software*, **159**, pág. 110432, URL <https://www.sciencedirect.com/science/article/pii/S0164121219302067>.

RESUMEN AUTOBIOGRÁFICO

Javier Iracheta Villarreal

Candidato para obtener el grado de
Doctorado en Ingeniería
Orientación en Tecnologías de la Información

Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica

Tesis:

DESCUBRIMIENTO DE RELACIONES ENTRE MICROSERVICIOS A
TRAVÉS DE UN GRAFO RELAJADO DE DEPENDENCIAS EXTRAÍDO
CON ANÁLISIS DE BITÁCORAS

Nací en la ciudad de Piedras Negras, Coahuila, México en el año de 1976. Obtuve el grado de Ingeniero en electrónica por el Instituto Tecnológico de Saltillo en el año 1998, también obtuve el grado de Maestría en Informática por la facultad de Administración y Contaduría Pública de la Universidad Autónoma de Nuevo León, en el año 2002, posteriormente obtuve el grado de Maestría en Administración por la universidad TEC Milenio, en el año 2012, en el año de 2019 empecé a estudiar el Doctorado en Ingeniería con Orientación a Tecnologías de Información en la Facultad de Ingeniería Mecánica y Eléctrica de la Universidad Autónoma de Nuevo León. Después de obtener mi grado de Ingeniero he estado más de 26 años laborando

continuamente en la iniciativa privada llevando proyectos de IT en empresas de múltiples giros y diferentes tamaños, actualmente estoy encargado del programa de calidad de software a nivel global de una compañía multinacional orientada a la construcción.