Bitácoras de la maestría, vol.VII

Aprovisionamiento ágil
Clasificación de malware
Optimización Giraph

Juan Felipe Gómez Manzanares
Álvaro Pachón de la Cruz
Sebastián Felipe Landínez García
Andrés Navarro Cadavid
César Loaiza
Gabriel Tamura Morimitsu

Editorial Universidad Icesi

BITÁCORAS DE LA MAESTRÍA

APROVISIONAMIENTO ÁGIL
CLASIFICACIÓN DE MALWARE
OPTIMIZACIÓN GIRAPH

BITÁCORAS DE LA MAESTRÍA

APROVISIONAMIENTO ÁGIL CLASIFICACIÓN DE MALWARE OPTIMIZACIÓN GIRAPH

Juan Felipe Gómez Manzanares
Álvaro Pachón de la Cruz
Sebastián Felipe Landínez García
Andrés Navarro Cadavid
César Loaiza
Gabriel Tamura Morimitsu

Editorial Universidad Icesi, 2020

Aprovisionamiento ágil – Clasificación de malware – Optimización Giraph

1 ed. Cali, Colombia. Universidad Icesi, 2020
252 p., 19x24 cm
Incluye referencias bibliográficas
ISBN: 978-958-5590-45-8 (PDF)
https://doi.org/10.18046/EUI/bm.7.2020

1. Agile computing 2. Cybersecurity 3. Optimization I.Tit
621.38 – dc22

Facultad de Ingeniería
Colección: Bitácoras de la maestría, vol. 7.

Rector: Francisco Piedrahita Plata
Decano Facultad de Ingeniería: Gonzalo Ulloa Villegas
Coordinador editorial: Adolfo A. Abadía

Producción y diseño: Claros Editores SAS.
Editor: José ignacio Claros V.
Traducción inglés - español: José Ignacio Claros V.
Impreso en Colombia / *Printed in Colombia.*

Editorial Universidad Icesi

Calle 18 #122-135 (Pance), Cali-Colombia
editorial@icesi.edu.co
www.icesi.edu.co/editorial
Teléfono: +57(2) 555 2334

La serie Bitácoras de la Maestría es una publicación de la Universidad Icesi que tiene como objetivo mejorar la difusión de los trabajos de grado meritorios de sus estudiantes, exponiéndolos a un público más amplio, no necesariamente académico, que pueda aprovecharlos en su cotidianidad. Se trata de "mover" las tesis, desde los anaqueles de las bibliotecas, hacia las manos de los actores de la vida diaria y establecer un vínculo entre autores y potenciales usuarios. Cuando el volumen se origina en la Facultad de Ingeniería, en cada volumen se incluyen tres trabajos con temática diversa. Por lo heterogéneo de su contenido, el nombre de cada volumen está compuesto por el nombre de la serie y el de los tres temas que incluye.

Juan Felipe Gómez Manzanares

Ingeniero Telemático y Máster en Informática y Telecomunicaciones de la Universidad Icesi (Cali, Colombia). Cuenta con más de cinco años de experiencia en la concepción, diseño, implementación y operación de infraestructura de TI *on-premise* y *cloud*. Actualmente se desempeña como ingeniero de infraestructura *cloud* y DevSecOps, principalmente con AWS, y es profesor de la Facultad de Ingeniería en la Universidad Icesi. Sus áreas de interés profesional incluyen: la implementación de arquitecturas *cloud* mediante infraestructura como código (IaC) –Terraform, Cloudformation–, *pipelines* de ingeniería continúa –CI/CD con Jenkins y GitActions–, micro-servicios –Docker y Kubernetes– y redes definidas por software. juanfe.2793@gmail.com

Álvaro Pachón de la Cruz

Profesor de tiempo completo y Jefe del Departamento de Tecnologías de la Información y las Comunicaciones de la Facultad de Ingeniería de la Universidad Icesi (Cali, Colombia). Es Doctor Ingeniero en Telemática y Máster en Ingeniería Telemática de la Universidad de Vigo (España); Especialista en Redes y Comunicaciones de la Universidad del Valle (Cali, Colombia) e Ingeniero de Sistemas de la Universidad Icesi. alvaro@icesi.edu.co

Sebastián Felipe Landínez García

Ingeniero en Electrónica y Telecomunicaciones de la Universidad del Cauca (Popayán, Colombia), destacado por su promedio académico con la medalla Francisco José de Caldas, y Máster en Informática y Telecomunicaciones de la Universidad Icesi (Cali, Colombia). Certificado en Redes Neuronales y Aprendizaje Profundo, y Machine Learning (Coursera). Como parte del programa de Jóvenes Investigadores e Innovadores de Colciencias, trabajó con el Grupo de Investigación en Informática y Telecomunicaciones (i2t) de la Universidad Icesi en el proyecto "Clasificador de malware para Android basado en redes neuronales artificiales. Su mayor área de interés profesional es el desarrollo de software. landinez@unicauca.edu.co

Andrés Navarro Cadavid

Profesor de planta y Director del Grupo de Investigación en Informática y Telecomunicaciones del Departamento de Tecnologías de la Información y las Comunicaciones de la Facultad de Ingeniería de la Universidad Icesi (Cali, Colombia). Es Doctor Ingeniero en Telecomunicaciones de la Universidad Politécnica de Valencia (España) y Máster en Gestión Tecnológica e Ingeniero en Telecomunicaciones de la Universidad Pontificia Bolivariana (Medellín, Colombia). Es presidente del capítulo Comunicaciones del IEEE Colombia y miembro de Grupo de Estudio 1 de la Unión Internacional de Telecomunicaciones. anavarro@icesi.edu.co

César Loaiza

Estudiante del PhD en Computer Science del University College Cork (Irlanda); es Ingeniero de Sistemas de la Universidad del Valle (Cali, Colombia) y Magister en Informática y Telecomunicaciones de la Universidad Icesi (Cali). Sus cuatro años de experiencia en proyectos incluyen: diseño e implementación de sistemas de big data y data analytics; desarrollo web; y solución de problemas computacionalmente complejos. Sus principales áreas de interés en investigación son: algoritmos distribuidos, machine learning, ciencia de redes y metaheurística. cdlq13@gmail.com

Gabriel Tamura Morimitsu

Doctor en Informática (Université Lille 1 Sciences et Technologies, Francia); Doctor en Ingeniería y Máster en Ingeniería de Sistemas y Computación (Universidad de Los Andes, Colombia); e Ingeniero de Sistemas y Computación (Universidad Javeriana, Colombia). Es profesor de tiempo completo de la Facultad de Ingeniería en la Universidad Icesi. Sus áreas de interés en investigación son: Context-Aware Self-Adaptive Software; gestión de tecnología e infraestructura informática; Model-Driven Software Development y Variability Modeling and Software Product Lines. Es IEEE Senior Member, Investigador Senior (Colciencias). gtamura@icesi.edu.co

Tabla de contenido

Presentación **23**

Modelo operativo para el aprovisionamiento ágil de aplicaciones y servicios de TI **25**

Resumen 27

Introducción 28

Marco teórico 31

Análisis de negocio 31

Red definida por software 32

Computación en la nube 33

DevOps 34

Experiencias relevantes 36

Desarrollo del proyecto 39

Infraestructura de servicios AWS y su gestión 40

Oportunidades de mejora en la infraestructura AWS 44

Definición e implementación de la centralización operación 57

Formalización de procesos del modelo operativo 63

Formalización de los procesos de desarrollo de requerimientos 71

Diseño del modelo operativo ágil 76

Implementación del modelo operativo ágil 89

Conclusiones 109
Referencias 111

Sistemas basados en RNA para clasificar malware en Android 117
Resumen 119
Introducción 120
Situación actual y antecedentes 120
La propuesta 122
Marco teórico 124
Aprendizaje de máquina 124
Android 134
Estado del arte 140
El proyecto 144
Metodología 144
Creación del dataset 145
Análisis preliminar 146
Herramientas de programación 147
Perceptrón multicapa 149
Evaluación 156
Conclusiones y trabajo futuro 156
Referencias 159

Búsqueda local distribuida basada en la exploración de vecindarios paralelos para el problema RDCMST 167
Resumen 169
Introducción 170
Marco teórico y estado del arte 173

Árboles de recubrimiento mínimo 173
Búsqueda local 175
Hadoop 176
Giraph 178
El modelo de computación Giraph 187
Un problema simple 187
Uso de las funciones avanzadas de Giraph 188
Estrategia de diseño global 193
Una búsqueda local iterada para resolver RDCMST 193
Solución completa basada en Giraph para resolver RDCMST 198
Evaluación 220
Fase de experimentos piloto 222
Instancias pequeñas 224
Instancias grandes 225
Análisis de resultados 228
Conclusiones y trabajo futuro 231
Referencias 233
Anexo 1. Descripción gráfica del movimiento completo 237

Índice de Tablas

Modelo operativo para el aprovisionamiento ágil...

Tabla 1. Fundamentos de las SDN **33**

Tabla 2. Servicios AWS usados por la empresa **40**

Tabla 3. Herramienta de diagnóstico y priorización: grupos de procesos **64**

Tabla 4. Herramienta de diagnóstico: grupos de proceso y sus procesos clave **65**

Tabla 5. Procesos clave priorizados **71**

Tabla 6. Planillas construidas con CloudFormation Designer **100**

Tabla 7. Recursos requeridos en AWS **105**

Sistemas basados en RNA para clasificar malware en Android

Tabla 1. Algoritmos utilizados para la detección de malware en Android **141**

Tabla 2. Comparativo entre propuestas del estado del arte **142**

Tabla 3. Permisos para datasets de malware y no malware **146**

Tabla 4. Permisos más solicitados para aplicaciones Android **147**

Tabla 5. Resultados del puntaje F promedio en el dataset de validación con arquitecturas de RNA **151**

Tabla 6. Resultados del puntaje F promedio en el dataset de prueba con arquitecturas de RNA con una capa oculta **152**

Tabla 7. Resultados del puntaje F en el dataset de validación con arquitecturas de RNA de dos capas ocultas **153**

Tabla 8. Resultados del puntaje F promedio en el dataset de prueba con s arquitecturas de RNA de dos capas ocultas **154**

Tabla 9. Resultados del puntaje F promedio en el dataset de validación con arquitecturas de RNA de tres capas ocultas **155**

Tabla 10. Resultados del puntaje F promedio en el conjunto de prueba con arquitecturas de RNA con tres capas ocultas **155**

Tabla 11. Comparativo de mejores resultados: sistema desarrollado / herramientas del estado del arte **157**

Búsqueda local distribuida basada en la exploración de vecindarios...

Tabla 1. Tamaño de los conjuntos de datos **223**

Tabla 2. Ejecución durante 30 minutos en pequeñas instancias aleatorias **224**

Tabla 3. Tiempo de ejecución de los experimentos **225**

Tabla 4. Resultados de los experimentos en términos de costo **225**

Tabla 5. Resultados usando varias soluciones iniciales **228**

Índice de Figuras

Modelo operativo para el aprovisionamiento ágil...

Figura 1. Portafolio y niveles de servicio **29**

Figura 2. Efecto de la ausencia de un modelo ágil en la operación **30**

Figura 3. Creación de infraestructura **42**

Figura 4. Gestión de la infraestructura **43**

Figura 5. Nuevo esquema de nomenclatura **49**

Figura 6. Etiquetado por proyecto **49**

Figura 7. Inventario AWS Config: Top 10 **52**

Figura 8. Arquitectura recomendada para la base de datos privada **53**

Figura 9. Arquitectura Base Datos Multi-AZ **55**

Figura 10. Infraestructura AWS definitiva **56**

Figura 11. Grupo de recursos SSM **59**

Figura 12. Iniciar sesión en EC2 con Session Manager **61**

Figura 13. Herramienta de diagnóstico **68**

Figura 14. Consolidado de la percepción del equipo de ingeniería **69**

Figura 15. Consolidado de la percepción de la gerencia **70**

Figura 16. Proceso "Nuevo requerimiento" **73**

Figura 17. Proceso "Asignación de un requerimiento" **74**

Figura 18. Proceso "Desarrollo de un requerimiento" **75**

Figura 19. Proceso “Asignación de una no conformidad” **76**
Figura 20. Proceso “Desarrollo de la solución de una no conformidad” **77**
Figura 21. Técnicas de priorización **81**
Figura 22. Valor vs riesgo **83**
Figura 23. Propuesta de tablero de priorización **84**
Figura 24. Pipeline de infraestructura **85**
Figura 25. Estructura de los repositorios de código fuente **86**
Figura 26. Inicio del pipeline de desarrollo **87**
Figura 27. Integración continua del pipeline desarrollo **88**
Figura 28. Última fase del pipeline de desarrollo **89**
Figura 29. Tarjeta Trello como historia de usuario **96**
Figura 30. Manejo final de las historias de usuario **96**
Figura 31. Arquitectura AWS CloudFormation **99**
Figura 32. Carga de plantilla en CloudFormation **100**
Figura 33. Parámetros de plantilla en CloudFormation **101**
Figura 34. Implementación infraestructura en AWS CloudFormation **102**
Figura 35. Eventos en CloudFormation **102**
Figura 36. Proceso de integración continua **105**
Figura 37. Configuración conexión repositorio CodeCommit **107**
Figura 38. Acciones posteriores a la tarea 1 **107**
Figura 39. Tarea para guardar artefacto en S3 **108**
Figura 40. Configuración despliegue desde Jenkins a CloudDeploy **109**
Figura 41. Pipeline en Jenkins **109**

Sistemas basados en RNA para clasificar malware en Android

Figura 1. Matriz de confusión **133**
Figura 2. Matriz de confusión (problema binario) **133**

Figura 3. Muestra de un AndroidManifest **135**

Figura 4. Muestra de descompilación de la herramienta APKTool **145**

Figura 5. Procedimiento ejecutado para probar el sistema desarrollado **150**

Búsqueda local distribuida basada en la exploración de vecindarios...

Figura 1. Ejemplo de una red óptica pasiva de largo alcance **171**

Figura 2. Modelo BSP **180**

Figura 3. Computación de un vértice que propaga el mayor valor recibido **181**

Figura 4. Mensajes reducidos por un combiner Max **183**

Figura 5. Arquitectura Giraph **184**

Figura 6. Ejecución de Giraph usando la función master compute para comunicar datos a través de agregadores **186**

Figura 7. Solución de un problema simple **187**

Figura 8. Superpasos para identificar la longitud promedio de la ruta hasta la hoja más lejana **191**

Figura 9. Operación de eliminación **195**

Figura 10. Escenarios para la operación de inserción **196**

Figura 11. Solución parcial al problema RDCMST **199**

Figura 12, Estrategia de eliminación factible **202**

Figura 13. Actualización de los atributos *f* después de la eliminación de un vértice **207**

Figura 14. Actualización de los atributos *b* después de la eliminación del vértice 1 **208**

Figura 15. Escenarios para la nueva hoja más lejana **209**

Figura 16. Localización para el vértice n3 **214**

Figura 17. Diagrama de despliegue **221**

Figura 18. Cantidad de hilos por trabajador realizando diez mil movimientos **223**

Figura 19. Ubicaciones evaluadas por minuto **227**

Acrónimos

Modelo operativo para el aprovisionamiento ágil...

AMI	Amazon Machine Image
API	Application Programming Interface
APM	Agile Project Management
AWS	Amazon Web Services
BGP	Border Gateway Protocol
CAPEX	Capital Expenditures
CD	Continuous Deployment
CDE	Continuous Delivery
CECAN	Centre for Evaluation of Complexity Across the Nexus
CI	Continuous Integration (Integración Continua)
CIDR	Classless Inter-Domain Routing
DMS	Database Migration Service
EBS	Elastic Block Storage
EC2	Elastic Compute Cloud
EIBT	Empresa Innovadora de Base Tecnológica
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
IAM	dentity and Access Manager
IoT	Internet of Things
IP	Internet Protocol
ITIL	Information Technology Infrastructure Library
LAMP	Linux Apache MySQL PHP
LISPS	Locator/Identifier Separation Protocol

NC	No Conformidad
OPEX	Operational Expenditures
OVSDB	Open vSwitch DataBase
P4	Programming Protocol-independent Packet Processors
PaaS	Platform as a Service
PBL	Problem Based Learning
PCEP	PCE Communications Protocol
QoS	Quality of Service
RA	Realidad Aumentada
RDS	Relational Database Service
RHEL	Red Hat Enterprise Linux
RV	Realidad Virtual
S3	Simple Storage Service
SaaS	Software as a Service
SDK	Software Development Kit
SDN	Software Defined Networks
SGSI	Sistema de Gestión de Seguridad de la Información
SMS	Server Migration Service
SNMP	Simple Network Management Protocol
SSH	Secure Shell
SSM	AWS SystemS Manager
TDD	Test-Driven Development
TI	Tecnologías de Información
TIC	Tecnologías Información y Comunicación
VPC	Virtual Private Cloud
VPN	Virtual Private Network
VSM	Value Stream Mapping
WAF	Web Application Firewall

Sistemas basados en RNA para clasificar malware en Android

.dex	Dalvid Executable
.odex	Optimized Dalvik Executable .
AI	Artificial Intelligence
AMGP	Android malware Genome Project

API	Application Programming Interface
APK	Android Application Package
ASUM	Analytics Solution Unified Method
CRISP-DM	Cross-Industry Standard Process for Data Mining
DBSCAN	Density-based spatial clustering of applications
ELF	Executable and Linkable Format
GPU	Graphics Processing Unit
IPC	Inter-Process Communications
KNN	K Nearest Neighbors
MDP	Markov Decision Process
ML	Machine Learning
RNA	Red Neuronal Artificial
RUP	Rational Unified Process
SEMMA	Sample, Explore, Modify, Model, Assess
SVM	Support Vector Machine
UID	Unique IDentifier
WORA	Write Once, Run Anywhere

Búsqueda local distribuida basada en la exploración de vecindarios...

ACO	Ant Colony Optimization
BKRUS	Bounded KRUSkal
BSP	Bulk Synchronous Parallel
ERDCMST	Edge-Disjoint RDCMST
HDFS	Hadoop Distributed File System
ICBLS	Iterated Constraint-Based Local Search,)
ILC	Iterative Learning Control
ILS	Iterated Local Search
LR-PON	Long-Range Passive Optical Network
LS	Local Search
MIP	Mixed Integer Programming
MPI	Message Passing Interface
MST	Minimum Spanning Tree
RDCMST	Rooted Distance-Constrained Minimum Spanning Tree
UDF	User Defined Function
VNS	Variable Neighborhood Search
YARN	Yet Another Resouce Negotiator

Presentación

Esta edición de Bitácoras de la Maestría es una buena muestra de la diversidad de problemas que se pueden resolver con el aporte de la informática. En el primer capítulo se trata tangencialmente un tema sobre el que hay mucha discusión, la industria 4.0, la denominada cuarta revolución industrial, con énfasis en "adaptación", la palabra clave para aprovechar sus bondades y eliminar el riesgo de sucumbir ante ella. En él se describe el proceso de transformación desarrollado bajo la filosofía *Agile* que se realizó en una empresa de base tecnológica que: provee servicios de TI, desarrolla software a la medida y facilita el despliegue de infraestructura en la nube de Amazon Web Service. El desarrollo del modelo operativo propuesto está enmarcado en el proyecto "Formación e innovación para el fortalecimiento de la competitividad del sector de las Tecnologías de la Información y Comunicación (TIC) de la región: FormaTIC e InnovaTIC Valle del Cauca" y busca ser un marco de trabajo de referencia para la adopción de prácticas ágiles en empresas de servicios de TI de esta región colombiana.

El tema del segundo capítulo es la seguridad de los dispositivos móviles que funcionan con un sistema operativo Android, es decir, de la seguridad del 85.4 % de los dispositivos que se usan en el mundo. En él se presenta el diseño, implementación y evaluación de un sistema de aprendizaje de máquina basado en redes neuronales artificiales, útil para la clasificación de *malware*, con un enfoque novedoso, no sobre características –que como la firma, hoy son "manejadas" con solvencia por los ciberatacantes–, sino sobre patrones de la aplicación ¿Relevante? Mucho, si se atiende a las estadísticas de detección de *malware*, que sitúan el nivel de éxito en cifras por debajo del 75 %; a la reconocida progresión de la capacidad técnica de los ciberatacantes, que les

permite evadir herramientas de detección tradicionales, cada vez con mayor facilidad; y al elevado costo de los perjuicios causados por los ciberataques, que fuentes reconocidas, como McAfee, valoran en alrededor de 600 billones de dólares al año.

El tercer capítulo presenta una solución para un problema de optimización complejo, el RDCMST (*Rooted Distance-Constrained Minimum Spanning Tree*), muy útil en temas con el diseño de redes de telecomunicaciones. Una revisión de investigaciones previas permitió concluir que aun cuando para el problema se han propuesto soluciones desde diversos enfoques, ellas tienen en común dos carencias importantes: no existen aproximaciones en paralelo que hayan sido diseñadas para aprovechar varias unidades de procesamiento y se limitan a instancias de unos pocos miles de vértices. El proyecto se enfocó entonces en la construcción de una estrategia distribuida para resolver el problema a partir de una búsqueda local que realiza una exploración paralela del vecindario, implementada en un software distribuido que permite lidiar con instancias del problema de cientos de miles de vértices. Para ello, debió afrontar retos asociados principalmente con el desempeño de la solución y el uso adecuado de los recursos computacionales disponibles. Una evaluación experimental de la estrategia desarrollada mostró su capacidad para manejar instancias de decenas de miles de vértices en un tiempo razonable y dio indicios de funcionar mejor con instancias más grandes y de tener un mejor desempeño bajo ciertas condiciones de la solución subóptima inicial en la búsqueda local.

José Ignacio Claros V.
Editor

Modelo operativo para el aprovisionamiento ágil de aplicaciones y servicios de TI

Juan Felipe Gómez Manzanares, MSc.
Álvaro Pachón De la Cruz, Ph.D

Citación

J. F. Gómez y Á. Pachón, "Modelo operativo para el aprovisionamiento ágil de aplicaciones y servicios de TI," en *Aprovisionamiento ágil – Clasificación de malware – Optimización Giraph* [Bitácoras de la maestría, vol. 7], Cali, Colombia: Universidad Icesi, 2020, pp. 25-115.

RESUMEN

El mundo se encuentra en su cuarta revolución industrial, la transformación digital de las organizaciones que ella trae consigo genera nuevos riesgos, a la vez que crea oportunidades de negocio; para aprovecharlas, las empresas deben transformar sus modelos –de negocio y operativos– a esquemas adaptables y flexibles que les permitan entregar valor a sus clientes en la medida en que lo requieran, en menos tiempo. En este documento se describe el proceso desarrollado en una empresa de base tecnológica, realizado siguiendo una metodología iterativa incremental que permite cambios durante su implementación, de acuerdo con las necesidades identificadas; su producto final es un modelo que reúne los valores y principios propios del desarrollo ágil. El trabajo fue realizado en una compañía ubicada en Cali (Colombia), cuya principal actividad es el desarrollo de servicios de tecnologías de información y software a la medida y el despliegue de infraestructura en la nube de Amazon Web Service.

INTRODUCCIÓN

Con la aplicación de las metodologías ágiles de desarrollo de software, se logra entregar software funcional en menor tiempo, lo que obliga a los sistemas que se encuentran en operación a adaptarse a las nuevas características del software de una manera igual de eficiente. Así, las empresas de desarrollo de servicios de Tecnologías de la Información [TI] se enfrentan a un nuevo reto: aprovisionar infraestructuras de prueba y producción para que sean flexibles y adaptables a las demandas de los clientes. El problema radica en que los equipos de operación no están establecidos para desplegar y aprovisionar servicios de TI de manera ágil, lo que resulta en cuellos de botella en el paso a producción del servicio y sus nuevas características [1].

El desarrollo del modelo operativo propuesto en este documento está enmarcado en el proyecto: "Formación e innovación para el fortalecimiento de la competitividad del sector de las Tecnologías de la Información y Comunicación (TIC) de la región: FormaTIC e InnovaTIC Valle del Cauca" [2] y busca ser un marco de trabajo de referencia para la adopción de prácticas ágiles en empresas de servicios de TI de esta región colombiana. El modelo se desarrolla con una metodología de trabajo iterativa e incremental y logra una construcción adaptativa del modelo; además, se aplican procesos formales de investigación que permiten la experimentación de nuevas tecnologías y la evaluación de su impacto durante el desarrollo del proyecto.

El trabajo fue desarrollado en una compañía de base tecnológica ubicada en Cali (Colombia), cuya principal actividad es el desarrollo de servicios de TI y software a la medida y el despliegue de infraestructura en la nube de Amazon Web Service [AWS]. De acuerdo con su misión, es una empresa de tecnología con soluciones y productos innovadores que generan valor a los clientes de una manera ágil, oportuna y flexible; según se expresa en su visión, espera estar en 2022 entre las primeras empresas de tecnología del Valle del Cauca, expandirse a nuevos mercados internacionales y ser reconocida como una empresa innovadora con clientes satisfechos.

Su enfoque comercial se basa en la construcción de relaciones de largo plazo con sus clientes, su estrategia para lograrlo se centra en cuatro valores: alianza con los clientes, innovación, flexibilidad y calidad. En su proceso comercial, escucha las necesidades de sus clientes, identifica una solución y luego la desarrolla.

Los servicios del portafolio de la empresa están clasificados en cuatro grupos: servicios consolidados, los más estables, totalmente desarrollados, con baja o nula necesidad de soporte técnico; servicios en consolidación, aquellos con un mayor nivel de innovación y poco tiempo en el portafolio, cuyo desarrollo se considera nuevo (aprovisionados para un máximo de dos clientes iniciales); servicios en desarrollo, los que se encuentran en fases avanzadas de desarrollo, en etapas de evaluación y ajustes de calidad para ser ofrecidos a los clientes; y servicios a la medida, aquellos que se desarrollaron para atender una necesidad especifica de un cliente en particular.

No todos los servicios del portafolio se ofrecen de manera directa, existen tres niveles: en el primero, se ubican servicios que se ofrecen directamente a los clientes; en el segundo, aquellos que soportan la operación de los de primer nivel y hacen posible la solución de los requerimientos del cliente; y en el tercero, los servicios de infraestructura, que soportan la operación de todos (Figura 1). Los servicios se relacionan entre sí a través del Virtual Connect, un concepto que garantiza soluciones modulares que se puedan adaptar a las necesidades de cada cliente.

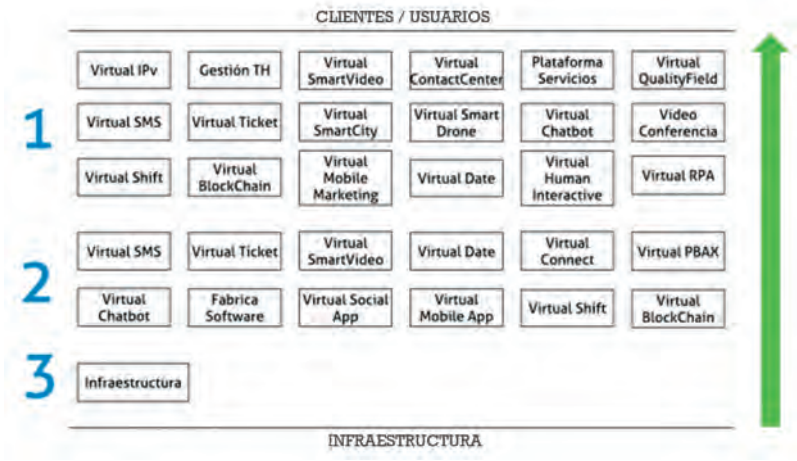


Figura 1. Portafolio y niveles de servicio

Originalmente, bajo el paradigma de desarrollo de software y de servicios de TI en cascada (secuencial), el ciclo de vida de desarrollo de sistemas de la empresa evolucionaba de forma predictiva, con la mayor parte de la

planificación por adelantado, desarrollando aplicaciones y servicios en una sola iteración, con un bajo o nulo involucramiento del cliente y con una entrega de valor tardía [3]; con la apropiación de ciclos de vida iterativos e incrementales se logró mejorar la parte del proceso encargada del desarrollo, no así la parte de la operación que quedó igual, con ciclos de vida estáticos [1].

En la Figura 2 se ilustra el problema actual que provoca la falta de un modelo ágil para la operación. En la parte superior se presenta un modelo predictivo en ambas partes –el esquema de trabajo viejo–, ahí los equipos de desarrollo entregan software completo para que los equipos de operaciones desplieguen y aprovisionen secuencialmente, sin que se presenten problemas en la entrega al cliente. En la parte inferior se ilustra lo que sucede con el nuevo esquema de trabajo, los equipos de desarrollo cuentan con modelos ágiles, que permiten la entrega de múltiples funcionalidades y características de una aplicación o servicio de TI de manera iterativa e incremental, pero los equipos de operaciones continúan desplegando y aprovisionando dichos servicios de manera secuencial, lo que provoca retrasos en la entrega de valor a los clientes.

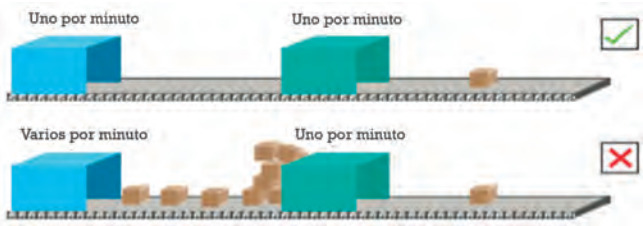


Figura 2. Efecto de la ausencia de un modelo ágil en la operación

Con el desarrollo de este proyecto se busca reducir el tiempo para asumir un servicio como consolidado, siguiendo un enfoque de aprovisionamiento ágil y adaptable, y aumentar así la competitividad de la empresa en el mercado. Se busca: reducir los tiempos de desarrollo y despliegue de los servicios de TI para la entrega de valor a los usuarios finales, optimizando la utilización de los recursos de TI, la gestión y el mantenimiento de los recursos (operación); y consolidar un modelo operativo ágil en el desarrollo y operación de los servicios de TI de la compañía. El modelo se desarrolla sobre la base de formalizar los procesos operativos, identificar oportunidades de mejora y apoyarse en diversas tecnologías emergentes para aumentar la agilidad de los procesos de operación

de la compañía. La ausencia de un modelo ágil ha tenido como efectos: mayores tiempos de entrega de valor y de nuevas características a los clientes; desaprovechamiento de los recursos humanos, financieros y tecnológicos de la empresa; pérdida de competitividad; e insatisfacción de los clientes.

Este proyecto se desarrolló con el objetivo de formular un modelo operativo para el despliegue, aprovisionamiento y gestión ágil de los servicios de TI para una Empresa Innovadora de Base Tecnológica [EIBT] que utiliza un modelo basado en un enfoque predictivo. Para alcanzarlo se definieron como objetivos específicos: construir un caso de negocio que justifique la formulación del modelo operativo como la mejor alternativa de solución; establecer un marco de trabajo para la gestión ágil de recursos y servicios de TI en un ambiente de computación en la nube; diseñar e implementar un modelo para el aprovisionamiento y la gestión de servicios de TI a través de prácticas ágiles; y validar el modelo operativo ágil de aprovisionamiento y gestión de servicios de TI en un caso de prueba, pequeño y controlado en la empresa.

MARCO TEÓRICO

Los conceptos clave para el desarrollo de este proyecto son: análisis de negocio, computación en la nube, redes definidas por software [*Software Defined Network*, SDN] y DevOps.

ANÁLISIS DE NEGOCIO

El análisis de negocio es la aplicación de conocimientos, habilidades y herramientas/técnicas para: determinar problemas y oportunidades; identificar necesidades de negocio y recomendar soluciones viables que las satisfacen; elicitar, analizar, especificar, comunicar y gestionar requerimientos; y definir beneficios y enfoques para medir y generar valor y analizar resultados [4]. En resumen, es el conjunto de actividades que permiten soportar la entrega de soluciones que agregan valor, resuelven una necesidad definida y están alineadas con los objetivos del negocio.

Realizar un análisis de negocio implica comprender el funcionamiento de una organización e identificar y caracterizar los procesos que siguen para llevar a cabo sus propósitos. En él se definen las capacidades que requiere la organización para proporcionar productos o servicios a sus clientes [5]; debe incluir: los objetivos estratégicos de la organización; su misión y visión;

el portafolio de proyectos y de servicios; la estructura organizacional y los interesados –externos e internos–; y las herramientas y procedimientos, con el fin de establecer cómo interactúan las unidades organizacionales y las partes interesadas dentro y fuera de la compañía [5].

El principal resultado de realizar un análisis de negocio es construir un caso de negocio a través del cual sea posible justificar la implementación de una solución definida que resuelva una necesidad o aproveche una oportunidad identificada en la organización. El proceso de construcción de un caso de negocio consta de seis fases principales: identificación del problema u oportunidad, valoración del estado actual, valoración del estado futuro o deseado, definición de alternativas de solución, hoja de ruta y ensamble del caso de negocio. Es importante destacar que este proceso define y describe las principales fases de desarrollo, pero no determina un cronograma explícito.

RED DEFINIDA POR SOFTWARE

El término fue originalmente acuñado y referenciado con *OpenFlow*, una propuesta de arquitectura de red donde el reenvío de los paquetes era responsabilidad de los equipos de comunicaciones, donde todos eran gestionados desde un componente central de control [6].

En una red IP [*Internet Protocol*] tradicional, cada dispositivo de comunicación tiene un plano de control, un plano de datos y un plano de gestión. Dichos dispositivos analizan cada paquete y realizan la conmutación de los paquetes decidiendo el próximo salto mediante tablas dependientes presentes en el plano de control [7]. Las SDN proponen una arquitectura diferente para una red: separa y consolida los planos de control de varios dispositivos de red en un solo sitio, donde el enrutamiento y las decisiones de calidad de servicio [*Quality of Service*, QoS] se toman utilizando una visión global de la red. En este nuevo paradigma, la red opera en términos de flujos de comunicación, no de flujos de paquetes [8].

Gracias a la separación del plano de control, con SDN es posible desarrollar software que pueda definir las operaciones de procesamiento y reenvío de manera centralizada para un grupo de recursos de red, a través de interfaces de programación abiertas y de un punto central de control, el controlador SDN [9]. El concepto de SDN se fundamenta en los principios descritos en la Tabla 1.

Tabla 1. Fundamentos de las SDN [10]

Principio	Descripción
Separación de los planos de control y gestión de los planos de reenvío y procesamiento	Permite la independencia entre el desarrollo y el despliegue de la red y con ello que el control y la gestión tengan funcionalidades propias, simplificando las características del reenvío y procesamiento. Adicionalmente, hace posible optimizar la gestión del ciclo vida del plano de datos y del plano de control, lo que implica una implementación ágil y flexible de las políticas que definan el comportamiento de la red.
Control centralizado de la red	Permite el uso eficiente de los recursos de la red debido a que se tiene una visión global de ella. Este principio se logra gracias a la implementación del controlador SDN, un componente lógico que puede abstraer los estados distribuidos de los recursos del plano de datos para formar una vista global de la red, sobre la cual las aplicaciones de control pueden programar la infraestructura de red subyacente.
Programabilidad de los servicios de red	Permite desarrollar aplicaciones de software que puedan programar las operaciones del plano de datos a través del controlador SDN. Esta plataforma centralizada de control habilita una interfaz de programación de aplicaciones [*Application Programming Interface*, API], conocida como la frontera norte, por medio de la cual las aplicaciones pueden intercambiar información con el controlador SDN para especificar las solicitudes de los servicios. Con este principio se logra un control ágil y flexible de los servicios ofrecidos por los recursos de red en el plano de datos. Además, permite a las aplicaciones expresar sus deseos de servicios de red (características de desempeño), dejando la configuración y optimización del uso de los recursos al controlador SDN.

En consecuencia, es posible automatizar los procesos de gestión y operación de una red reduciendo hasta un 30 % los gastos operativos [OPEX, *Operational Expenditures*] y un 80% el gasto de capital [CAPEX, *Capital Expenditures*] de la infraestructura de red; no es de extrañar entonces que los líderes del sector de las Tecnologías de la Información y las Comunicaciones [TIC] hayan adoptado este enfoque, convirtiendo la infraestructura de red en un recurso estratégico para el negocio, permitiendo así optimizar y acelerar el ciclo de vida de las aplicaciones [11].

COMPUTACIÓN EN LA NUBE

La computación en la nube [*Cloud Computing*] establece los mecanismos a través de los cuales se realiza la entrega bajo demanda de: poder de computo, almacenamiento, bases de datos y aplicaciones, entre otros recursos de TI, a través de plataformas de servicios digitales, vía Internet y navegadores web; esto implica que los usuarios y organizaciones no requieren hacer inversiones en

dispositivos de hardware ni invertir tiempo en la gestión del *hardware*, reduciendo CAPEX y OPEX [12]. Los proveedores de nube gestionan los componentes principales (i.e., servidores de aplicaciones, red, almacenamiento) a partir de centros de datos ubicados en diferentes locaciones alrededor del mundo, cada uno con diferentes tecnologías y servicios de vanguardia, que buscan ayudar a los usuarios y organizaciones a alcanzar sus objetivos corporativos y requerimientos de tecnología bajo un modelo de pago por consumo [13]. El concepto, inicialmente percibido como riesgoso y confuso, ha evolucionado a una estrategia organizacional fundamental para pequeñas y grandes empresas, un elemento crucial de sus estrategias de entrega de servicios [14].

Los servicios *cloud* pueden ser organizados en tres categorías principales de acuerdo con el nivel de control y responsabilidad del usuario en la configuración del servicio [13]:

- *Infraestructure as a Service* [IaaS], donde el usuario se encarga de gestionar el servidor virtual, el sistema operativo, las bases de datos y la seguridad en el acceso a la aplicación, mientras que los componentes físicos son responsabilidad del proveedor de la nube;
- *Platform as a Service* [PaaS], donde el proveedor se encarga de manejar el *hardware*, los sistemas operativos, la seguridad en el acceso y las bases de datos, mientras que el cliente se concentra en el desarrollo y despliegue de sus aplicaciones y se despreocupa de la gestión de la infraestructura subyacente; y
- *Software as a Service* [SaaS], donde el proveedor del servicio pone a disposición de los clientes su propio software y el usuario solo se preocupa por utilizarlo.

DevOps

El término nace de la unión de los equipos de desarrollo (Dev) y operaciones (Ops), aunque fue inicialmente mencionado en 2009 [1], no fue sino hasta 2013 cuando empezó a ganar popularidad [15]. Si bien no cuenta con una definición universalmente aceptada, si existen elementos clave para una: el primero, es el Manifiesto Ágil [16], un acuerdo suscrito entre varios empresarios del mundo del desarrollo tecnológico, quienes definieron cuatro valores y doce principios. De este manifiesto se origina la mentalidad ágil como núcleo del concepto DevOps [3].

DevOps puede ser considerado como un tipo de movimiento ágil presente en compañías de desarrollo de software y servicios de TI, cuyo objetivo es reducir la distancia entre los equipos de desarrollo y operación, tradicionalmente rígidos y rigurosos en los procesos, a través de la automatización del aprovisionamiento y configuración de infraestructura utilizando arquitecturas definidas por software, el cual busca además mejorar la comunicación, colaboración y confianza de ambos equipos, encontrando el balance entre innovación y estabilidad y entre velocidad de entrega y calidad del servicio [1].

Para empresas de alto rendimiento –como Amazon–, DevOps constituye una combinación de filosofías culturales, prácticas y herramientas que incrementan la capacidad de una organización para proporcionar aplicaciones y servicios de TI a gran velocidad, para así desarrollar y mejorar productos con mayor rapidez que las organizaciones que utilizan procesos tradicionales de desarrollo de software y administración de infraestructura. Esta velocidad permite a las organizaciones servir mejor a sus clientes y competir de forma más eficaz en el mercado [16]. Adicionalmente, Amazon define a la retroalimentación de los clientes y usuarios como elemento fundamental de una cultura DevOps, habilitando canales de comunicación efectivos y definiendo un ciclo de retroalimentación que permite entregar al cliente justo lo que necesita.

Las prácticas continuas –integración continua, entrega continua y despliegue continuo–, constituyen un elemento fundamental en la definición de DevOps, están destinadas a ayudar a las organizaciones a acelerar su desarrollo y entrega de funciones de software sin comprometer la calidad, ofreciendo los siguientes beneficios: obtener comentarios rápidos del proceso de desarrollo de software y de los clientes; tener lanzamientos frecuentes y confiables, que conducen a una mejor satisfacción del cliente y calidad del producto; y fortalecer la conexión entre los equipos de desarrollo y operaciones, eliminando las tareas manuales [17].

La integración continua [*Continuous Integration*, CI] es una práctica de desarrollo ampliamente establecida en la industria de desarrollo de software en la cual los miembros de un equipo integran y fusionan el trabajo del equipo de desarrollo, lo que le permite a las compañías de software: tener un ciclo de lanzamiento más corto y frecuente, mejorar la calidad del software y aumentar la productividad de sus equipos [17].

La entrega continua [*Continuous Delivery*, CDE] tiene como objetivo garantizar que una aplicación una vez que pasa con éxito las pruebas automatizadas y los controles de calidad, está lista para producción, para lo que emplea un conjunto

de prácticas tales como CI e infraestructura como código [*Infraestructure as Code*, IaC] para entregar software a un entorno productivo [17].

El despliegue continuo [*Continuous Deployment*, CD] va un paso más allá, implementando la aplicación de forma automática y continua en entornos de producción o de clientes. Su objetivo es implementar de manera automática y constante cada cambio en el entorno de producción [17], lo que implica que cada desarrollo que termina pasa por un proceso de entrega continua y es desplegado automáticamente en producción.

EXPERIENCIAS RELEVANTES

En la revisión bibliográfica previa al desarrollo del proyecto se identificaron cuatro estudios que demuestran la importancia de implementar modelos operativos ágiles, al evaluarlos es posible apreciar cómo dichos modelos operativos permiten mejorar el desempeño de los equipos de trabajo, independiente del sector dónde se ejecuten. Cabe resaltar que cada uno basa su modelo en una práctica ágil diferente.

La optimización de las tareas y los procesos a través de una metodología iterativa e incremental, que permite segmentar problemas complejos en porciones más simples, es un elemento común en estos estudios. Con estas tareas simples, un equipo trabaja de manera más eficiente y entrega resultados, valor, productos o servicios de TI en menos tiempo, satisfaciendo las necesidades del cliente o usuario.

En el primero de ellos, Benítez [18] describe la adaptación de una metodología esbelta [*Lean Manufacturing*] a los procesos productivos de una empresa de ingeniería dedicada a la elaboración de proyectos de construcción de subestaciones eléctricas. Describe un modelo esbelto como la base fundamental para lograr una operación flexible y adaptativa.

Lean Manufacturing consiste en una serie de principios, conceptos y técnicas diseñadas para eliminar el desperdicio y establecer un sistema de producción eficiente, que permita realizar entregas de los productos requeridos a los clientes: cuando son requeridos, en la cantidad requerida y sin defectos. Sus modelos se centran en la generación de valor al cliente, entendiendo valor como todo aquello que hace que se cumplan las funcionalidades esperadas por el cliente, con un nivel de calidad, costo y plazo esperados y por lo cual está

dispuesto a pagar. Este concepto de valor es clave para la implementación de modelos operativos ágiles pues forma parte de los doce principios incluidos en el Manifestó Ágil [3].

Benítez [18] muestra cómo la eficiencia productiva de la organización mejora con el análisis de la cadena de valor realizado utilizando la herramienta mapa de flujo de valor [*Value Stream Mapping*, VSM]. Este mapa le permite obtener una perspectiva general del conjunto –no sólo de cada proceso–, y mejorar todo el proceso productivo buscando disminuir los desperdicios de recursos y tiempo. Bajo este modelo, la compañía logró: optimizar el flujo de materiales, procesos e información a lo largo del proceso productivo; entregar valor al cliente; y reducir al máximo los desperdicios de la organización.

El segundo estudio se enfoca en S4N, una empresa colombiana desarrolladora de software fundada en 2008 con el propósito declarado de "construir los mejores productos de software del mundo para transformar las organizaciones y mejorar la vida de la gente" [19].

Los cambios en la cultura y en la forma de trabajo de S4N para adaptarse a las prácticas ágiles de desarrollo de software se presentan en [20], donde se describe el trabajo desarrollado para convertirla en una compañía que se adapta a las necesidades del cliente y entrega software funcional en menos tiempo, incrementando la colaboración entre los equipos de desarrollo y operaciones, haciendo más eficientes los procesos y permitiendo entregas rápidas y seguras.

Al adoptar un enfoque de trabajo ágil, en S4N cada miembro del equipo/proyecto cuenta con el conocimiento y las herramientas necesarias para acercar las arquitecturas de software de la empresa al nivel de las organizaciones de alto rendimiento, desarrollando procesos de despliegue automatizados que agilizan las entregas y generan valor con mayor rapidez [20]. El modelo operativo actual de S4N cuenta con tres procesos principales: CI, CDE, y CD.

En el tercer estudio, Senabre-Hidalgo [21] analiza el proceso de adopción de un marco de trabajo ágil en el CECAN [Centre for Evaluation of Complexity Across the Nexus], un centro de investigación de ciencias sociales y políticas del Reino Unido, y demuestra cómo se logra mejorar la colaboración, aumentar la eficiencia y obtener resultados en menor tiempo. En su trabajo, identifica las principales características y herramientas de *Scrum* que contribuyen a la gestión colaborativa y a la coordinación eficiente de las tareas en los procesos de investigación social.

La mayoría de los modelos operativos ágiles tratan de minimizar el riesgo de ejecución de un proyecto desarrollando software por iteraciones e incrementos, pero dejan de lado los componentes sociales de un equipo de trabajo. El autor propone un modelo centrado en la gestión ágil de proyectos [*Agile Project Management*, APM], un *framework* que: soporta progresos continuos e incrementales según las prioridades de trabajo, se adapta a los cambios, se centra en el trabajo en equipo, se enfoca en los aspectos sociales del desarrollo del proyecto y apoya la cocreación entre integrantes del equipo a través de grupos multifuncionales. Es claro que la adopción de métodos ágiles permite visualizar y compartir las tareas en progreso, maximizando así las posibilidades de éxito en proyectos complejos en ambiente multidisciplinares

Senabre-Hidalgo mostró las ventajas de utilizar el enfoque propuesto por APM para el trabajo en equipo y la selección de algunos elementos de *Scrum* en la coordinación eficiente de tareas, logrando disminuir la brecha entre investigación, puesta en práctica y publicación de resultados. Las principales ventajas al implementar el proyecto fueron: mejor manejo de la complejidad de los proyectos; capacidad de contar con un equipo autogestionado; mayor flexibilidad y reducción de la burocracia en los procesos; y adaptabilidad a los cambios.

El cuarto estudio se refiere a la "IBM Garage Field Guide" [22], en ella se propone la adaptación a través de talleres dónde se diseñan y construyen productos mínimos viables que resuelven las necesidades de negocio y se escalan fácilmente a través de la compañía. Dice IBM que la agilidad inicia con la transformación de la cultura y la mentalidad de los equipos y sus líderes, adoptando tecnologías disruptivas que ayudan a construir nuevas plataformas de negocio entregando valor, para lo que propone tres valores cruciales: el cambio cultural, para trabajar con CI y CDE; la adopción de las mejores prácticas, reconociendo que la evolución de un ciclo de vida en cascada a un ciclo continuo requiere de la combinación de esas mejores prácticas con la experiencia y el conocimiento propios; y la construcción en la nube, lo que implica modernizar las aplicaciones existentes y crear aplicaciones nativas para la nube usando las plataformas públicas, privadas o multinube. Su metodología de adaptación para habilitar los principios ágiles incluye siete elementos fundamentales, la cultura –el centro de ellos, el principal–, y seis acciones: descubrir, visualizar, desarrollar, reflexionar, operar y aprender, enmarcados en el ciclo cooperar–cocrear–coejecutar.

DESARROLLO DEL PROYECTO

El desarrollo del proyecto inicia con la construcción de un caso de negocio, con él: se identifica una oportunidad de mejora crucial, junto con sus causas y efectos; se presenta una propuesta de solución a la problemática planteada; y se establecen los parámetros de un estado futuro deseado. Al construir el caso de negocio se define el problema, los objetivos y el contexto y se justifica la ejecución del proyecto. La construcción del caso de negocio contempla seis etapas: identificación del problema u oportunidad, valoración del estado actual, valoración estado futuro o deseado, definición de alternativas de solución, hoja de ruta y ensamble del caso de negocio.

Al iniciar el proyecto se realizaron entrevistas con personas claves de la empresa –gerentes, directora de proyectos y líder técnico–, útiles para conocer la misión, visión, estructura jerárquica, servicios ofrecidos y operación de la compañía, e identificar la ausencia de un portafolio de servicios formal. Cada interesado apelaba a su memoria para definir y describir cada servicio ofertado por la compañía, con lo que se obtenían diferentes descripciones de un mismo servicio y se percibían dificultades para determinar las relaciones con otros servicios.

Se creó entonces un portafolio de servicios formal para la compañía, el cual define, describe, clasifica y relaciona cada uno de sus servicios; de él, se realizaron cinco versiones, las cuales fueron validadas con cada uno de los interesados para garantizar la inclusión de la perspectiva de cada uno y la construcción de un portafolio unificado con el que todos estuvieran de acuerdo. Este proceso fue crucial para que la compañía determinará sobre qué servicios enfocar su crecimiento.

Como se dijo, no todos los servicios del portafolio se ofrecen de manera directa a los clientes, sino que existen tres niveles de servicio: oferta directa, soporte e infraestructura. Esta clasificación le permitió a la empresa identificar los servicios sobre los que debía tener un desarrollo más fuerte en sus componentes de cara al cliente [front-end], los de nivel 1, por su interacción directa con el usuario, y aquellos sobre lo que debía tener un enfoque fuerte en los componentes *back-end*, los de niveles 2 y 3.

Todos los servicios se relacionan entre sí a través del *Virtual Connect* –un concepto que manejaba inicialmente el gerente principal de la compañía y que

con el desarrollo del portafolio se fue compartiendo y consolidando con todo el equipo de trabajo–, el cual se refiere a la conexión de todos los servicios y la garantía de provisión de soluciones modulares que se puedan adaptar a las necesidades de cada cliente.

INFRAESTRUCTURA DE SERVICIOS AWS Y SU GESTIÓN

Como siguiente paso se propuso la elaboración de un documento de definición de la infraestructura de AWS de la empresa, la idea era describir los componentes de AWS utilizados en el despliegue y operación de su infraestructura y describir la arquitectura de dos servicios, especificando cada uno de los elementos de la infraestructura AWS que los componen y especificando sus relaciones.

La nube AWS cuenta con 1.017 características y servicios distribuidas en 19 categorías de servicio diferentes [23]: computación; almacenamiento; base de datos; migración; redes y entrega de contenido; herramientas para desarrolladores; herramientas de administración; servicios multimedia; seguridad, identidad y conformidad; análisis; aprendizaje automático; servicios móviles; Realidad Aumentada [RA] y Realidad Virtual [RV[; integración de aplicaciones; interacción con clientes; productividad empresarial; *streaming* de aplicaciones y escritorios; IoT [*Internet of Things*]; y desarrollo de videojuegos.

De esas características y servicios, la empresa utiliza veintitrés servicios para el despliegue y operación de sus servicios, los cuales se clasifican, según su forma de interacción, en: constantes, aquellos que la organización utiliza en su operación diaria; a la medida, los que se usan para resolver una necesidad especifica de un cliente; y exploratorios, aquellos que el equipo de desarrollo está investigando con el fin de agregar nuevas funcionalidades a los servicios ya existentes o desarrollar nuevos. En la Tabla 2 se presentan los servicios de cada categoría (su descripción está disponible en [24]).

Tabla 2. Servicios AWS usados por la empresa

Categoría	Servicios
Constantes	Elastic Compute Cloud [EC2], Lambda, API Gateway, Simple Storage Service [S3], S3 Glacier, Relational Database Service [RDS], Route 53, CodeCommit, Workdocs e Identity and Access Management [IAM].
A la medida	Dynamo DB no SQL, Virtual Private Cloud [VPC], Rekognition, Web Application Firewall [WAF] y Shield.
Exploratorios	Red Shift, Cloud Front, Database Migration Service [DMS], Server Migration Service [SMS], Lex, Device Farm, Machine Learning y Workspace.

En el modelo propuesto, no basta con definir y clasificar los servicios utilizados, es necesario que ellos cuenten con una arquitectura clara, para lograr migrar a un enfoque de trabajo sin afectar la operación e identificar oportunidades de mejora de dicha arquitectura. Para una muestra de la arquitectura de algunos servicios de la empresa se puede consultar [25, secc. 8.3.1]

La infraestructura de la empresa tiene como base tres VPC: una con su nombre, Virtual Video y una con el nombre de su mayor cliente, en ellas se realiza el despliegue de sus veintidós servicios. Cada servicio cuenta con dos instancias EC2, una para producción, otra para desarrollo y pruebas. Además, cada servicio tiene una base de datos relacional y un espacio de almacenamiento de archivos en el S3 y cuenta con sus propias reglas de entrada, definidas a través de un grupo de seguridad. Todos estos componentes son gestionados por una sola persona, el líder técnico.

Al iniciar el desarrollo de un nuevo servicio, el líder técnico crea una nueva instancia EC2 con sistema operativo Red Hat Enterprise Linux [RHEL], la que se aprovisiona con las configuraciones por defecto de AWS; posteriormente, el líder técnico ingresa por *Secure Shell* [SSH] a la instancia e instala y configura los paquetes necesarios para habilitar el desarrollo del servicio o la aplicación. Los paquetes necesarios hacen referencia a: servicios web (e.g., Apache, PHP); intérpretes de código (e.g., Python, Node.js); kits de desarrollo de software [*Software Development Kits*, SDK] (e.g., el SDK de AWS, Java, Python); servicios de conectividad (i.e., herramientas de red, como NetTools); y motor de base de datos (MySQL o DynamoDB). En algunas ocasiones se realiza la configuración de la base de datos dentro de la instancia EC2.

Finalizada la configuración de la instancia, se crea una AMI [Amazon Machine Image] con la que se clona la instancia para el ambiente de desarrollo y pruebas, y se genera un respaldo para que en caso de fallas, sea posible regresar a un estado estable. Las instancia son aprovisionadas en alguna de las tres VPC definidas. Durante el aprovisionamiento de la nueva instancia, el líder técnico procede a configurar el grupo de seguridad para permitir el tráfico hacia los puertos específicos del servicio.

A continuación, el líder técnico configura la base de datos. En este punto, la base de datos del servicio puede ser desplegada localmente en la instancia o en el RDS de AWS; la decisión depende de la complejidad de la base de datos: si se trata de pocos datos, el despliegue se hace local, sino se hace en el RDS. El líder técnico configura los accesos a la base de datos, las tablas iniciales

y sus relaciones y crea el *bucket* en S3 para el almacenamiento de archivos (i.e., imágenes y documentos). En ocasiones, el líder técnico debe configurar funciones *lambda* para ejecutar tareas muy específicas, él define cual será el lanzador de la función y dónde será enviada la respuesta. Una vez finaliza la configuración descrita, el líder técnico comparte la información de conexión a cada servicio desplegado con el equipo de desarrollo. El resumen del proceso descrito se presenta en la Figura 3.

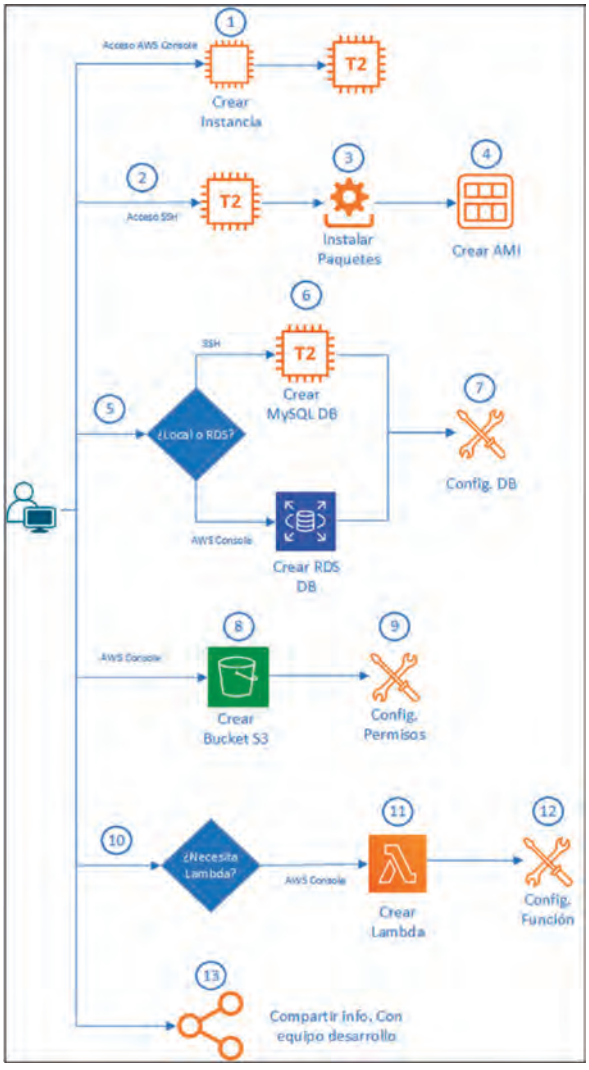


Figura 3. Creación de infraestructura

El líder técnico es también el encargado del mantenimiento de la infraestructura desplegada, en consecuencia debe: actualizar los paquetes de software a las versiones más estables, resolver los errores reportados por el equipo de desarrollo, apoyar al equipo de soporte en la resolución de fallos reportados por los usuarios y configurar permisos y servicios adicionales, entre otras tareas de la operación diaria de cada servicio. A esto se suma la ausencia de monitoreo del estado de los servicios y su infraestructura. La Figura 4 corresponde al esquema del modelo de gestión de infraestructura de la empresa.

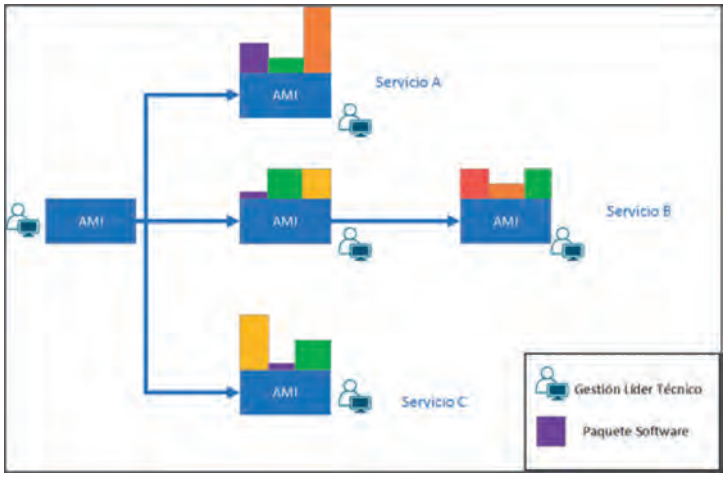


Figura 4. Gestión de la infraestructura

Es evidente entonces que gestionar los servicios en la empresa es un trabajo complejo, con un nivel de automatización muy bajo –sino nulo–, lo que provoca una sobrecarga en el líder técnico y dificulta los procesos de desarrollo, innovación y operación de los servicios. Esta situación afecta a los clientes, quienes experimentan lentitud en el uso de los servicios, fallos en las aplicaciones, problemas de compatibilidad, brechas de seguridad y demoras en la entrega de nuevas características. Cabe resaltar que la calificación del servicio de soporte es de 98 % y que la mayoría de los casos se resuelve en un primer contacto. Por lo tanto, no todos los problemas son resueltos por el líder técnico, porque en la mayoría de los casos, no se trata de problemas de infraestructura, sino que son problemas que pueden resolverse con programación u opciones de configuración.

OPORTUNIDADES DE MEJORA EN LA INFRAESTRUCTURA AWS

Una vez establecidos los procesos de gestión de la infraestructura de los servicios de la empresa en AWS, se identificaron Oportunidades de Mejora [OM] sobre dicha infraestructura en: arquitectura formal AWS, optimización de costos, seguridad, tolerancia a fallos, monitoreo, automatización y buenas prácticas arquitectura, como se describe a continuación. Para definir cada oportunidad se contó con el apoyo de AWS Trusted Advisor, un servicio en línea que en tiempo real ayuda con la provisión de recursos según las prácticas recomendadas por AWS [26].

OM1. Arquitectura formal AWS

La primera oportunidad implementada fue la creación de un diagrama de la arquitectura actual de la empresa en AWS, el cual permite que los equipos de desarrollo e infraestructura y los directivos pueden comprender dónde y cómo se encuentran desplegados los servicios de la compañía. Con él fue posible: explicar y justificar cada una de las mejoras que serán implementadas; establecer una estrategia comercial que muestra a los clientes la alta disponibilidad manejada en la empresa; y eliminar una VPC que no agrega valor a su modelo de negocio.

Las VPC se diagramaron usando Lucidchart, un espacio de trabajo visual que combina diagramas, visualización de datos y colaboración, y cuenta con un plugin que permite importar la arquitectura actual de AWS a la plataforma y generar un primer diagrama de ella. Esta primera aproximación se puede ajustar para obtener la arquitectura completa. Se caracterizó un total de 294 recursos de AWS, todos necesarios en la operación de la empresa. El detalle completo de la infraestructura puede consultarse en [27].

OM2. Optimización de costos

La revisión de los servicios de AWS utilizados por la compañía permitió identificar ahorros potenciales que se logran con la implementación de las siguientes oportunidades:

- *Optimización de instancias EC2 reservadas*. Basados en las recomendaciones de AWS es posible migrar algunas instancias EC2 dinámicas a instancias con recursos de computo reservados. Se configuraron instancias reservadas de los servicios más estables de la empresa, es decir de aquellos que no

mostraban patrones de crecimiento considerables. Cabe mencionar que las instancias reservadas ofrecen una tarifa horaria con descuento y una reserva de capacidad opcional para las instancias de EC2, y su facturación proporciona un descuento de hasta el 75 % en comparación con el precio de las instancias bajo demanda

- *Instancias EC2 con baja utilización*. Dado que es posible realizar un proceso de parada e inicio [*Start/Stop*] de las instancias en los horarios con cero demanda del servicio, se identificaron las instancias de baja utilización y los horarios en los cuales la demanda es mínima. Al analizar el total de instancias desplegadas, se encontró que catorce de las diecinueve instancias EC2 analizadas utilizan menos del 1 % de su CPU. Luego de la preparación de un listado de instancias elegibles para un proceso de este tipo, todas las instancias de prueba pasaron a un proceso *Start/Stop*. En este tipo de procesos, las direcciones IPv4 privadas se conservan, pero las direcciones IPv4 públicas, se liberan cuando se detiene la instancia, y se les asigna una nueva dirección IPv4 pública cuando se reinicia; lo mismo ocurre con el nombre de dominio público creado al momento de iniciar la instancia, se elimina al ejecutar el stop en la instancia EC2 y se genera uno nuevo en su reinicio. Asimismo, cuando se tiene una instancia con ClassicLink, una vez reiniciada la instancia se debe ligar nuevamente a la VPC. Es posible utilizar una IP elástica para evitar el problema del cambio en direcciones públicas y tener una IP pública estática.
- *Direcciones IP elásticas sin uso*. Se identificaron cuatro direcciones IP elásticas no asignadas a una instancia EC2. Dado que AWS realiza un cargo nominal por dirección IP, sin importar si está o no asociada a una instancia EC2, se asignaron a los servidores de prueba para garantizar su uso continuo.
- *Volúmenes Elastic Block Storage [EBS] sin uso*. Inicialmente se identificaron ocho volúmenes posiblemente sin uso; luego de su identificación efectiva, se realizó un *snapshot* para generar un respaldo y se procedió con su eliminación. Cabe mencionar que los cargos a los EBS se mantienen mientras el volumen este activo, esté o no asociado a una instancia EC2.

OM3. Seguridad

Se identificaron siete alertas de seguridad como oportunidades de mejora:

- *Snapshots públicos*. Se identificó un *snapshot* de un volumen EBS marcado como público, esto implica que todas las cuentas de AWS y cualquier usuario

pueden acceder a la información contenida en él. Esta configuración se había realizado originalmente para un respaldo de un cliente antiguo, pero nunca retornó a su estado de volumen privado. Se realizó el cambio.

- *Rotación de la llave de acceso IAM*. Se rotó esta llave para evitar el acceso no deseado a los recursos de la cuenta. Se crearon cuatro grupos de usuarios con sus respectivos permisos de acceso, se configuró un usuario para cada persona del equipo de la empresa y se asignaron los usuarios a los grupos, de esta manera, se evita que todas las personas entren con la cuenta *root* de AWS. Asimismo, los usuarios destinados al uso de las SDK o las API tienen las políticas específicas para su uso.
- *Política de contraseñas*. Se realizó un *upgrade* de la política de contraseñas, la cual obliga a los usuarios a fortalecerlas incluyendo una mínimo de seis caracteres, una letra mayúscula y un número.
- *Autenticación multifactor*. Aunque para la cuenta root AWS recomienda activar la autenticación multifactor para evitar suplantaciones, la compañía decidió no implementar esta mejora.
- *Revisión de las políticas de los grupos de seguridad*. La recomendación es especificar los puertos específicos a los que tienen acceso los usuarios, con base en ella se revisó cada grupo de seguridad y se configuraron los puertos específicos de acceso.
- *Permisos globales de los buckets S3*. Algunos de estos *buckets* tienen permisos de acceso global, luego de su revisión se determinó cuáles deben mantener dicha configuración, para garantizar la correcta operación del servicio. Se conservaron los permisos públicos a aquellos utilizados para la generación de reportes y la carga de imágenes.
- *Actualización del agente EC2 en instancias Windows*. Para estas instancias es necesario realizar la actualización manual del agente EC2, este permite realizar actualizaciones de software y aumenta la seguridad de la instancia. La empresa decidió no actualizar estos componentes por los problemas de compatibilidad con el servicio desplegado que puede provocar la actualización automática del sistema operativo o de software instalado en las maquinas Windows.

OM4. Tolerancia a fallos

Se identificaron las siguientes oportunidades para aumentar la tolerancia a fallos:

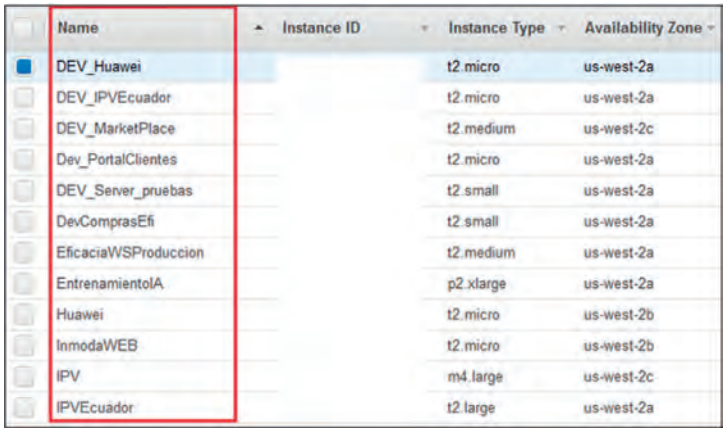
- *Actualizar EBS Snapshots*. Es importante realizar *snapshots* de seguridad de los volúmenes de almacenamiento EBS, ellos son un respaldo de la información almacenada en cada volumen. Inicialmente, estos *snapshots* se realizaron manualmente para garantizar que todo volumen tenga su respectivo respaldo; luego, este proceso se automatizó con la herramienta AWS Systems Manager [SSM].
- *Balance de zonas de disponibilidad*. La zona de disponibilidad us-west-2 mostraba once instancias en estado operativo, se recomendó distribuir la operación de esas instancias en diferentes zonas de disponibilidad para evitar afectaciones en el servicio si una zona de disponibilidad presenta fallas, sin embargo, la empresa decidió no migrarlas porque existe una alta disponibilidad y no se desea tener un balance mayor.
- *Versionamiento y ciclos de vida para los buckets S3*. Los treinta y cinco *buckets* activos de S3 no cuentan con el manejo de versiones de archivos que les permitiría recuperarse de fallos en la aplicación o de acciones de usuarios mal intencionados. En este caso, no se implementó el versionamiento, pero sí los ciclos de vida de los archivos.

  En los *buckets* de S3 con información de clientes y usuarios que no hacen parte de la empresa se configuró una regla de ciclo de vida que envía: los datos que no han sido consultados por un periodo de hasta tres años, al servicio de S3 de acceso poco frecuente; y aquellos que no han sido accedidos por más tiempo, al servicio de Amazon Glacier, dónde quedan archivados. Esta política le permite a la empresa continuar almacenando los datos a un costo mucho menor.
- *Redundancia en túneles VPN [Virtual Private Network]*. AWS recomienda que cada conexión VPN cuente con dos túneles para permitir una conexión redundante en caso de fallas o mantenimiento. La configuración de un túnel adicional no genera cargos adicionales. Esta VPN se encuentra definida entre la VPC de un cliente y su red privada, para lograr un canal encriptado entre la nube de AWS y el cliente. Se configuró efectivamente el segundo túnel VPN y se logró la redundancia esperada.
- *Configuración del Auto Scaling*. El servicio de auto escalamiento permite realizar la configuración y escalamiento automatizado de los recursos de AWS, principalmente de instancias EC2; su uso se recomienda para evitar afectaciones en la oferta de los servicios. En este caso no se realizó esta configuración porque las aplicaciones son *stateful*.

OM5. Monitoreo y gestión

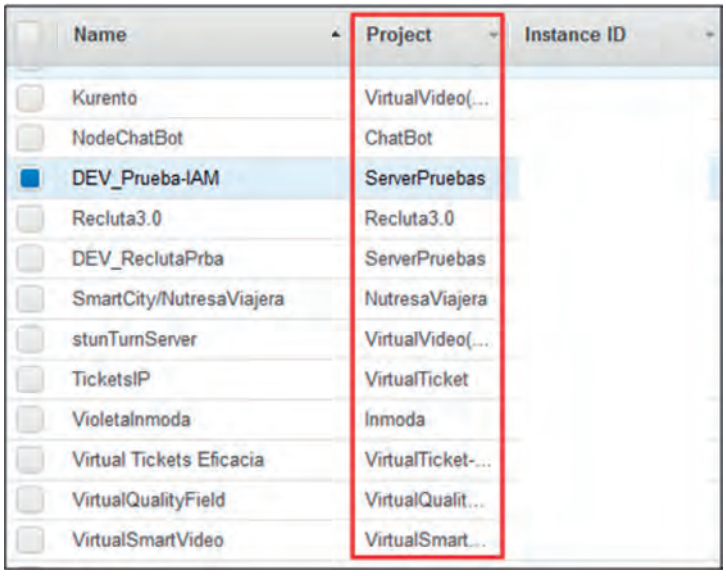
En este aspecto, se identificaron las siguientes oportunidades de mejora:

- *Logging con AWS CloudTrail.* Esta función, útil para el manejo de los logs de los servicios de AWS, que provee visibilidad completa de la actividad en la cuenta de AWS, no se encontraba activa, por lo que se procedió con su habilitación, para así tener un seguimiento completo de toda la actividad dentro de la cuenta de AWS de la compañía. Adicionalmente, se crearon cuatro trails para tener trazabilidad completa de cuatro servicios de la empresa: Video Quality Field, Iniciativas MAC, IPvPruebas e IPvColombia.
- *Bucket S3 logging.* Se identificaron treinta y cuatro *buckets* del servicio de S3 sin las funcionalidades de *logging* activadas. Con el almacenamiento de los *logs* es posible identificar de manera detallada cada solicitud y la información consultada del *bucket* S3. Se implementaron *logs* para las actividades de escritura y consultas en los *buckets* de S3, para contar con un registro de la información almacenada en ellos. Cabe mencionar que habilitar el registro de acceso a un *bucket* de Amazon S3 no genera cargos extra.
- *Nomenclatura de los recursos AWS.* La nomenclatura utilizada por la empresa en los recursos de AWS (EC2, S3, RDS, VPC, entre otros) dificulta la identificación del propósito de cada servicio y su relación con la infraestructura. Al menos el 90 % de los recursos solo podía ser identificado con el id único de recurso, un número hexadecimal con formato: i-0459e337fd8179a62. Que no tengan un nombre claro dificulta las tareas de mantenimiento porque hace necesaria la búsqueda uno a uno hasta encontrar el recurso adecuado –y aun así, existe la posibilidad de errores de elección–. Una vez reconocido el recurso por actualizar, modificar o instalar nuevos paquetes de software, se procede con el mantenimiento, se identifica cada uno de los recursos dentro de la infraestructura y se nombra de manera clara utilizando etiquetas *Name*, las cuales diferencian los recursos de producción de los de prueba incluyendo "*DEV*" antes del nombre del recurso (ver Figura 5). Adicionalmente, se creó una etiqueta de proyecto, con ella se agrupan todos los recursos asociados a un proyecto o servicio específico dentro de la cuenta de AWS de la empresa. Luego, es posible agrupar dichos recursos por un elemento en común (e.g., por proyecto, como aparece en la Figura 6) y configurar políticas de automatización en conjunto.



Name	Instance ID	Instance Type	Availability Zone
DEV_Huawei		t2.micro	us-west-2a
DEV_IPVEcuador		t2.micro	us-west-2a
DEV_MarketPlace		t2.medium	us-west-2c
Dev_PortalClientes		t2.micro	us-west-2a
DEV_Server_pruebas		t2.small	us-west-2a
DevComprasEfi		t2.small	us-west-2a
EficaciaWSProduccion		t2.medium	us-west-2a
EntrenamientoIA		p2.xlarge	us-west-2a
Huawei		t2.micro	us-west-2b
InmodaWEB		t2.micro	us-west-2b
IPV		m4.large	us-west-2c
IPVEcuador		t2.large	us-west-2a

Figura 5. Nuevo esquema de nomenclatura



Name	Project	Instance ID
Kurento	VirtualVideo(...	
NodeChatBot	ChatBot	
DEV_Prueba-IAM	ServerPruebas	
Recluta3.0	Recluta3.0	
DEV_ReclutaPrba	ServerPruebas	
SmartCity/NutresaViajera	NutresaViajera	
stunTurnServer	VirtualVideo(...	
TicketsIP	VirtualTicket	
VioletaInmoda	Inmoda	
Virtual Tickets Eficacia	VirtualTicket-...	
VirtualQualityField	VirtualQualit...	
VirtualSmartVideo	VirtualSmart...	

Figura 6. Etiquetado por proyecto

- *Paneles CloudWatch*. Si bien Amazon CloudWatch ya era utilizado para monitorear el estado de algunas instancias EC2, esta actividad no estaba organizado en paneles por proyecto. Se recomendó configurar CloudWatch para monitorear los recursos de AWS y gestionar las alarmas generadas por los recursos. Con CloudWatch es posible automatizar algunas acciones

en el momento cuando aparezca una alarma. Se configuraron dieciocho paneles de CloudWatch para el monitoreo integral de cada proyecto, en cada uno de ellos se puede visualizar de manera rápida la utilización de la CPU de las instancias asociadas al proyecto y el tráfico de entrada y de salida. Cada gráfica puede verse de manera detallada, obteniendo los registros específicos en una escala de tiempo definida por el usuario. La herramienta permite al usuario seleccionar la escala de tiempo para la consulta de las estadísticas recopiladas (i.e., una, tres y doce horas; un día, tres o una semana). Con la configuración de estos paneles, se tiene la visión holística de los proyectos y pueden identificar cuáles son los de mayor interacción con los usuarios y las instancias EC2 que se están quedando cortas de recursos, antes de que se presenten quejas de los usuarios o se experimenten fallas en la operación normal del servicio.

La última tarea importante relacionada con este servicio fue la creación y configuración de once alarmas de comportamiento, estas alarmas están asociadas a una única métrica de CloudWatch. La alarma realiza una o varias acciones cuando la métrica alcanza un umbral determinado por el usuario durante varios períodos de tiempo. La acción puede ser una acción de Amazon EC2, una acción de Amazon EC2 Auto Scaling o una notificación enviada a un tema de Amazon SNS. En este caso, las alarmas configuradas tienen dos propósitos principales: monitorear la utilización de la CPU de algunas instancias EC2 y monitorear los túneles VPN entre la VPC y la red local de su principal cliente. Las alarmas centradas en la utilización de la CPU validan que ella no supere el 90 % en un periodo de cinco minutos, si dicho umbral se supera, la alarma reinicia toda la instancia e informa a los administradores de la infraestructura de AWS; por su parte, las alarmas centradas en la conexión VPN validan el estado de la conexión, si este es diferente a 1 dentro de un periodo de cinco minutos, se alerta al equipo de la empresa y al responsable de la conexión en el lado del cliente. Con las alarmas, el equipo de infraestructura de la empresa ya no debe estar pendiente todo el tiempo del monitoreo del estado de las instancias y puede delegar dicha responsabilidad en los servicios de AWS.

- *Trazabilidad con AWS Config.* Una vez se ha logrado efectuar la revisión de la infraestructura, es importante efectuar la implementación del servicio AWS Config, para examinar, auditar y evaluar las configuraciones de los recursos AWS de la compañía. AWS Config monitorea y registra constantemente la configuración de sus recursos de AWS a través de la

configuración de reglas de automatización; con esta herramienta se pueden revisar los cambios en las configuraciones y las relaciones entre los recursos de AWS y profundizar en los historiales detallados de configuración de recursos, de esta manera, se simplifican: el trabajo de auditoría, los análisis de seguridad, la administración de cambios y la resolución de problemas operativos [28]. En el desarrollo de este proyecto se utilizaron solamente dos reglas de conformidad y una de inventario y monitoreo. Las reglas de AWS Config, son reglas personalizables y predefinidas que AWS Config utiliza para evaluar si los recursos de AWS se ajustan o no a las prácticas recomendadas.

Las dos reglas de conformidad configuradas son: ec2-instance-managed-by-systems-manager y ebs-optimized-instance, ambas programadas para ejecutarse cada 24 horas: la primera comprueba si las instancias de Amazon EC2 son administradas por SSM y solo muestra las instancias que no se encuentren administradas por él; la segunda comprueba la optimización de los volúmenes EBS, ella ofrece el mejor rendimiento para estos volúmenes porque reduce al mínimo la contención entre las operaciones de entrada y salida (I/O) de Amazon EBS y otro tráfico procedente de la instancia, es decir, optimiza la velocidad de transferencia de almacenamiento entre 425 Mbps y 14000 Mbps [29].

La regla de inventario y monitoreo del servicio AWS Config permite el conteo de los recursos activos en la plataforma de AWS y su categorización, para mostrarlo a través de un panel centralizado. Dicho inventario se lista de manera detallada y permite revisar los eventos ocurridos en cada recurso. En la Figura 7 se puede apreciar el inventario resumido por categorías de recurso en AWS para la empresa, en un Top 10 de los recursos más utilizados en la cuenta. En el inventario desde AWS Config, donde es posible seleccionar cualquiera de los recursos listados y ver de manera detallada su información) De la información ahí consignada se destaca la línea de tiempo de configuraciones, un servicio que permite ver el historial de modificaciones del recurso de una manera sencilla, en ella se especifican los cambios y eventos del recurso y es posible evaluar cada cambio auditando todos los comandos ejecutados y el usuario o función que lo realizó. De esta manera, el personal técnico encargado de la infraestructura de AWS cuenta con un seguimiento más claro de lo que ocurre en la infraestructura y puede dar un mejor soporte a los clientes cuando se presenta alguna eventualidad. La implementación de

Figura 7. Inventario AWS Config: Top 10

esta mejora permite identificar de manera rápida a qué nivel se presentan los fallos –software o infraestructura– y tener la trazabilidad completa de los recursos.

OM6. Buenas prácticas de diseño de arquitectura

Un correcto diseño de la arquitectura es fundamental para lograr la utilización óptima de AWS y garantizar la continuidad del negocio y la mayor disponibilidad para los clientes. La herramienta Well-Architecture Framework de Amazon le ayuda a los arquitectos que trabajan con la nube a crear la infraestructura más segura, resistente, eficaz y de alto rendimiento posible para sus aplicaciones. Las oportunidades de mejora para la empresa en este aspecto son:

- *Base de datos en subredes privadas*. De acuerdo con las recomendaciones de AWS, las bases de datos del servicio RDS se deben ubicar en una subred privada que no esté expuesta a Internet; con la subred privada solo se manejan direcciones IP conocidas dentro de ella y no pueden ser enrutadas por fuera de ella. Para acceder a la información contenida en RDS, los

usuarios deben acceder a través de una instancia de la aplicación que será la única con los permisos necesarios, con el uso de un grupo de seguridad, para consultar, guardar, modificar o borrar información de la base de datos. Este diseño brinda una capa de seguridad adicional a los datos almacenados en la base y evita ataques de terceros desde la Internet. Para lograr este esquema es necesario agregar un servicio de NAT Gateway que se encargue de comunicar las subredes públicas –dónde están las instancias de la aplicación–, con las subredes privadas. En la Figura 8 se puede apreciar la arquitectura descrita.

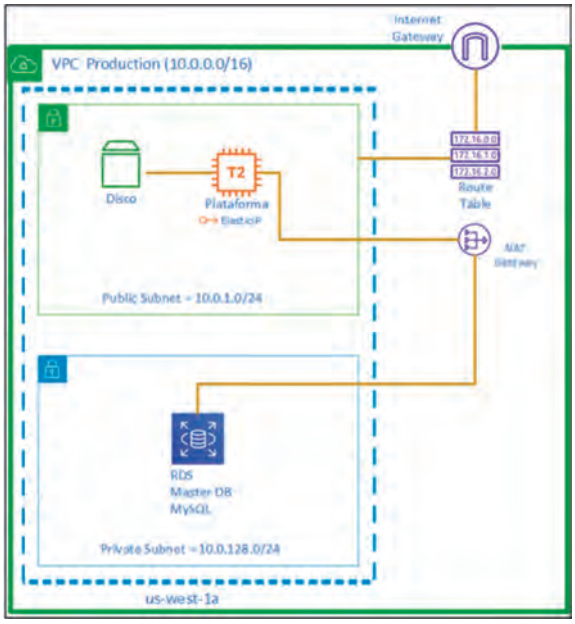


Figura 8. Arquitectura recomendada para la base de datos privada

La empresa decidió no implementar esta recomendación porque considera que afecta la normal operación de los servicios de la compañía, principalmente, del servicio de IPv. La dificultad primordial radica en los usuarios móviles del servicio IPv, quienes utilizan un servicio de sincronización para subir los datos recolectados en su día laboral a la plataforma, pues dicho servicio envía directamente la información a la base de datos para reducir los saltos y mejorar el rendimiento. Sin embargo, se estableció el compromiso de realizar el cambio en el desarrollo de la

aplicación para evitar que el dispositivo móvil se conecte directamente a la base de datos a través de Internet. Cabe resaltar, que los datos se envían cifrados desde el dispositivo móvil a RDS.

Una vez realizados los cambios en el código de la aplicación, se realizó la migración a tres subredes privadas, una por cada zona de disponibilidad, cada una con un *Classless Inter-Domain Routing* [CIDR] privado en el rango de IPsv4 172.31.16.0/20, 172.31.0.0/20, 172.31.32.0/20. Adicionalmente, se asociaron dos grupos de seguridad con sus respectivas reglas de entrada y salida.

- *Implementación de bases de datos en múltiples zonas (multi-AZ) de disponibilidad.* Esta es una de las principales recomendaciones de Amazon y busca aumentar la integridad de los datos con una réplica de lectura en diferentes zonas de disponibilidad, de tal manera que si ocurre un fallo en la base de datos principal, la información se encuentra protegida en otra zona de disponibilidad, lo que reduce al mínimo el riesgo de pérdida de información. Cuando se aprovisiona una instancia de base de datos Multi-AZ, Amazon RDS crea automáticamente dos instancias, una de base de datos principal y otra de reserva, en donde se replican sincrónicamente los datos en una zona de disponibilidad diferente. En caso de que se produzca un fallo en la infraestructura, Amazon RDS lleva a cabo una conmutación automática por error [30].

  En la Figura 9 se muestra la arquitectura de AWS con una implementación Multi-AZ. Esta implementación se puede apreciar a través del servicio RDS, en la verificación del despliegue en múltiples zonas de disponibilidad.

- *Producción y desarrollo/pruebas en diferentes VPC.* Una buena práctica de AWS es separar los ambientes de producción y desarrollo/pruebas en VPC diferentes, esto permite una separación desde la arquitectura, que logra un ambiente de *sandbox* –un entorno de pruebas aislado para desarrollo y validación de los servicios antes de su despliegue en producción–. Inicialmente se consideró utilizar una VPC que estaba sin uso para crear ese ambiente de pruebas, sin embargo, la decisión final fue borrarla y crear una nueva "desde cero", con una única subred. A ella se migraron las instancias de prueba.

  En AWS no es posible mover una instancia de una VPC a otra "en caliente", sino que se debe programar una ventana de mantenimiento. Para esto, se ingresa a cada instancia y se efectúa una revisión de los servicios y

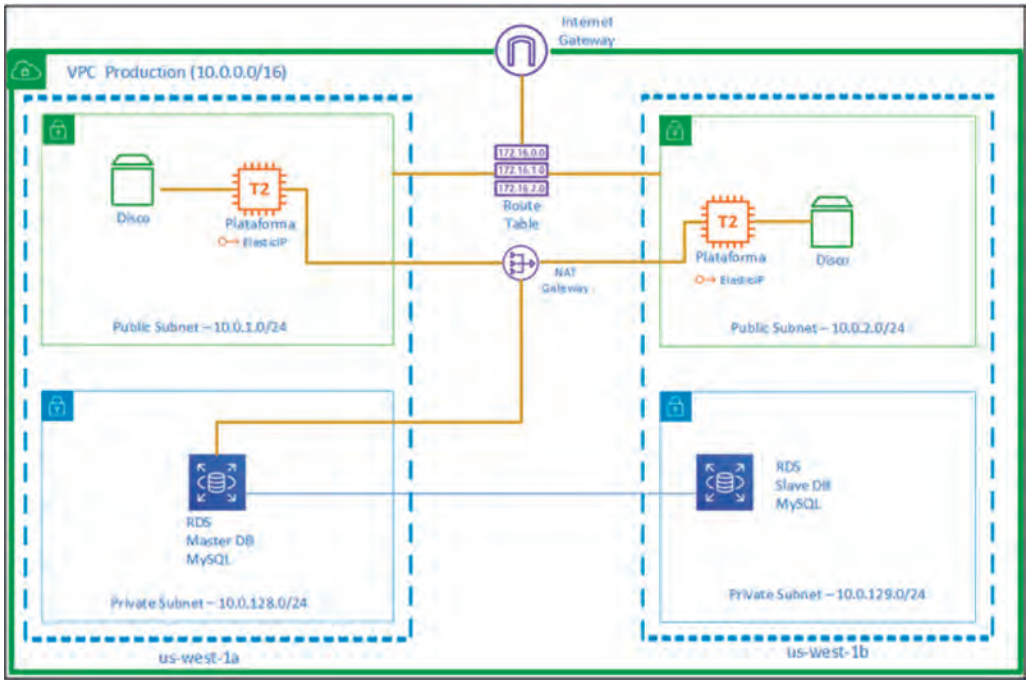


Figura 9. Arquitectura de la base de datos Multi-AZ

procesos en ejecución, para así tener un inventario de la actividad de la instancia; posteriormente, se configura la instancia instalando algunos paquetes adicionales, como una base de datos MySQL local. La empresa determinó que la VPC de pruebas no tendría el servicio de RDS, sino una base de datos local.

Una vez culminada la configuración de la instancia, se creó una AMI que sería utilizada en la migración y se lanzó una nueva instancia que toma como referencia dicha AMI en la VPC de desarrollo/pruebas;

Finalmente, se validó la correcta ejecución de esta instancia en su nuevo ambiente, a través de una serie de pruebas programadas con los equipos de infraestructura y desarrollo. La migración culminó con la asignación de la IP pública a esta nueva instancia y la eliminación de la instancia anterior. Este proceso fue realizado con cada una de las instancias de desarrollo/pruebas de cada servicio del portafolio de la empresa.

Con las oportunidades de mejora implementadas la infraestructura logra un nuevo estado: estable, optimo, seguro, escalable y con trazabilidad en toda la actividad de la cuenta. Esta nueva infraestructura mejora la operación de la compañía; reduce las tareas de soporte, mantenimiento y respaldo; y agrega valor al negocio porque permite crear nuevas ofertas comerciales basadas en una infraestructura segura, eficiente y disponible, que garantiza la integridad de la información.

En la Figura 10 se puede apreciar la arquitectura completa de la empresa, se ha simplificado el diagrama generalizando la cantidad de instancias desplegadas en cada VPC.

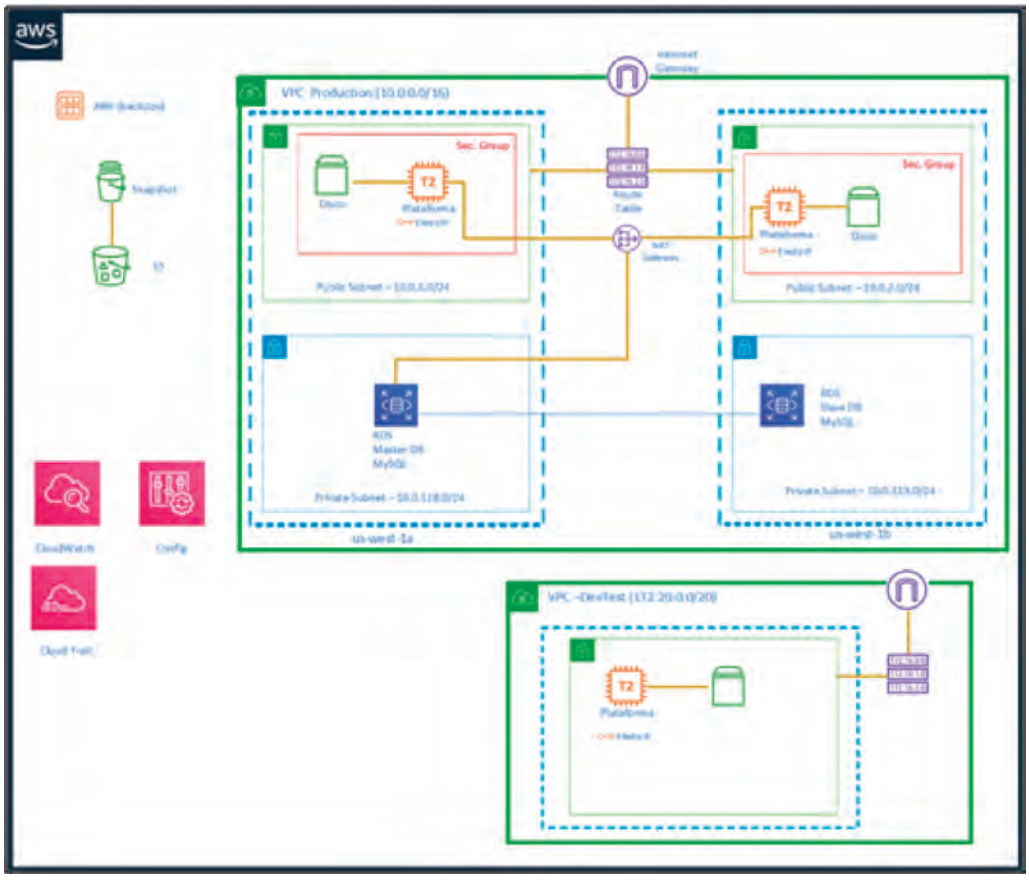


Figura 10. Infraestructura AWS definitiva

DEFINICIÓN E IMPLEMENTACIÓN DE LA CENTRALIZACIÓN OPERACIÓN

Con la implementación de las oportunidades descritas en la sección anterior, se evidencia una mejora en la operación de los servicios actuales y una reducción en los costos operativos, sin embrago, el modelo operativo de gestión de la infraestructura no presenta cambios, la gestión de la infraestructura continúa realizándose manualmente, el líder técnico crea y realiza el mantenimiento de toda la infraestructura. Para solucionar este problema, se plantea la necesidad de centralizar las labores de gestión de la infraestructura a través de una herramienta web simple que permita la automatización de tareas de mantenimiento y evite gestionar cada recurso de la cuenta de AWS de manera independiente. Las alternativas de solución son: migrar todos los servicios a entornos PaaS, implementar un controlador SDN e investigar por servicio interno en AWS. Cada una de ellas se analizó para determinar la mejor.

En la primera opción, migrar hacia un entorno de plataforma como servicio, la gestión de la infraestructura la asume el proveedor de la nube, así pues AWS configuraría las VPC, subredes, instancias y demás componentes, y la empresa solo tendría que preocuparse por el desarrollo del software y el despliegue de los servicios. Esta alternativa se descartó porque esa migración implica efectuar cambios en la arquitectura de cada uno de los servicios y diseñarlos y desarrollarlos de nuevo, para lograr su despliegue en ambiente PaaS. Además, al ser AWS quién configura y gestiona los recursos, los costos se elevarían y con ello se perdería lo ahorrado con las oportunidades de mejora.

La segunda opción propone implementar y configurar un controlador SDN para gestionar de manera centralizada los recursos de la cuenta de la empresa en AWS. Esta solución es viable, la comunicación entre el controlador y ciertos recursos de AWS se hace a través de la RESTFul API, lo que simplifica la comunicación. Para probar esta opción, se seleccionó el controlador OpenDaylight, una plataforma abierta modular para personalizar y automatizar redes de cualquier tamaño y escala, diseñada como base para soluciones comerciales que abordan una variedad de casos de uso en entornos de red existentes. Una de sus mayores ventajas es el uso de una arquitectura de microservicios –cada uno un protocolo o servicio que se quiere habilitar dentro de la instalación del controlador–, al que es posible agregarle servicios adicionales y con ello nuevas funcionalidades. OpenDaylight soporta una amplia y creciente gama de protocolos de red, tales como: OpenFlow, P4, BGP

[Border Gateway Protocol], PCEP [PCE Communications Protocol], LISP [Locator/Identifier Separation Protocol], OVSDB [Open vSwitch Database] y SNMP [Simple Network Management Protocol] [31].

La instalación y configuración de OpenDaylight en una instancia EC2 dentro de la cuenta AWS de la empresa no presentó dificultades; además, se validó que el controlador SDN puede gestionar una red desde AWS. Sin embargo, la gestión de los recursos de AWS con los paquetes disponibles por defecto no fue posible, de hecho, sería necesario desarrollar un nuevo módulo usando P4 [*Programming Protocol-independent Packet Processors*], algo que está por fuera del alcance del proyecto. Por esta razón, se descartó esta opción.

Para valorar la tercera opción se contactó a un consultor en Amazon con la idea de conocer si existe un servicio que permita la gestión centralizada de la infraestructura de AWS; él recomendó utilizar AWS Systems Manager [SSM], herramienta que brinda un panel de control centralizado de la infraestructura en AWS y permite tanto ver los datos operativos de varios servicios de AWS como la automatización de tareas de mantenimiento en todos los recursos de AWS. Con él es posible agrupar los recursos de cada aplicación, por ejemplo, instancias de Amazon EC2, *buckets* de Amazon S3 o instancias de Amazon RDS. Adicionalmente, permite el análisis de datos de la operación de la cuenta para monitoreo y solución de problemas, actuando directamente sobre los grupos de recursos.

En resumen, SSM simplifica la administración de las aplicaciones y los recursos, agiliza la detección y resolución de problemas operativos y facilita el uso y la administración de la infraestructura de manera segura [32]. Por lo dicho, esta es la herramienta seleccionada para centralizar la gestión de los recursos. Para su implementación se llevó a cabo el siguiente proceso:

- *Configuración de la etiqueta de proyecto en cada recurso AWS*. Las etiquetas permiten que los recursos se pueden agrupar según el proyecto y el servicio.
- *Creación del grupo de recursos por proyecto*. Este es un modo de agrupamiento lógico de los recursos asociados a una carga de trabajo concreta. Los grupos se pueden crear, actualizar o eliminar mediante programación con API o a través del panel de configuración web. Con la herramienta Resource Groups, se organiza y consolida la información según el proyecto y los recursos que utilice, lo que facilita el monitoreo y la gestión de la infraestructura de AWS por proyecto. En la Figura 11 se aprecia la correcta configuración de cada uno de los grupos de recursos.

Figura 11. Grupo de recursos SSM

- *Instalación de un agente SSM en instancias EC2*. Para automatizar la gestión de la infraestructura AWS es necesario instalar un agente del SSM en cada una de las instancias EC2 que se encarga de procesar las solicitudes desde el servicio SSM en la nube y configura la máquina tal y como se especifique en la solicitud; el agente del SSM devuelve la información del estado y de la ejecución al servicio SSM mediante el servicio de mensajería de EC2. La instalación se realizó de acuerdo con lo previsto en [33, pp. 74-93].
- *Configuración de los roles de administración y automatización*. Los roles en AWS permiten asignar políticas de seguridad a las interacciones entre servicios de AWS. Se deben crear dos que autoricen al servicio del SSM para modificar y acceder a los recursos que se van a gestionar de manera centralizada: el primero, el rol de gestión de instancias EC2, habilita los permisos para que el servicio del SSM pueda consultar y modificar las instancias EC2 de la empresa; y el segundo, el rol de automatización, otorga permisos a los servicios de automatización del SSM para modificar recursos de la empresa en AWS. El primer rol permite monitorear, configurar y realizar labores de mantenimiento. Debido a que las rutinas de automatización se ejecutan siempre según la configuración definida, sin necesidad de aprobación del usuario, es importante dar los permisos necesarios a cada rutina para evitar generar errores y problemas en la operación.
- *Rutina de actualización del agente SSM*. Para garantizar el mejor funcionamiento del agente SSM y evitar problemas en la ejecución de las

tareas automáticas, se configura una rutina de automatización desde el State Manager encargada de actualizar el agente SSM a su versión más reciente, la cual se ejecuta semanalmente, cada domingo.

- *Rutina de inventario de recursos AWS*. Aprovechando que las instancias EC2 se encuentran asociadas con el SSM, se realiza la configuración de una rutina de automatización que se ejecuta cada hora, la cual genera el inventario de software de todas las instancias EC2 asociadas a él, lo que facilita conocer las configuraciones del sistema y las aplicaciones instaladas. Con el inventario es posible recopilar datos sobre aplicaciones, archivos, configuraciones de red, servicios de Windows, registros, roles de servidor, actualizaciones y otras propiedades del sistema; así, es posible conocer qué ejecuta cada instancia y cuál es el estado de la operación del servicio y así efectuar la actualización del inventario en la opción de *insights* en el SSM. En el inventario se puede observar: el software que se está ejecutando, los roles del servidor, las direcciones IP, los sistemas operativos y las aplicaciones.
- *Integración SSM–CloudTrail*. Se hace para tener la visibilidad de los *logs* generados por los recursos de AWS desde la consola de administración del SSM. Analizar los *logs* desde esta consola le permite al administrador filtrar por grupo de recursos, según el proyecto que se desee analizar.
- *Integración SSM–Config*. Con ella es posible tener una trazabilidad centralizada de todos los cambios que se realicen en la infraestructura de AWS, es posible también filtrar la información por grupo de recursos.
- *Integración SSM–CloudWatch*. Se realiza para visualizar los principales paneles de monitoreo configurados en CloudWatch desde la consola central de administración del SSM y así monitorear desde ella a los principales recursos utilizados por la empresa en su operación.
- *Configuración del Session Manager dentro del SSM*. Una actividad frecuente en la empresa es el acceso a sus servidores a través de SSH para realizar tareas de mantenimiento, instalación de nuevos componentes y corrección de errores. Para agilizar dicha labor, se configura la herramienta Session Manager dentro del SSM. El Session Manager es una funcionalidad de SSM completamente administrada que permite gestionar instancias de Amazon EC2 a través de un shell interactivo basado en navegador con un solo clic; el Session Manager proporciona una administración de instancias segura y auditable sin necesidad de abrir los puertos de

entrada o administrar claves SSH. En consecuencia, los administradores de la empresa no deben guardar las llaves y credenciales de acceso a cada instancia, sino seleccionar la instancia desde el SSM e iniciar la sesión. Para garantizar que el servicio de SSM pueda iniciar sesión con la consola de administración, se deben agregar permisos adicionales al rol de EC2.

El siguiente fragmento de código muestra la configuración de la política incrustada, con ella, al ir a la ventana del Session Manager y seleccionar Start Session se muestran las instancias asociadas al SSM (Figura 12), de donde se puede seleccionar la que se desee conectar, e iniciar sesión:

```
---
Version: '2012-10-17'
Statement:
- Effect: Allow
 Action:
 - ssm:UpdateInstanceInformation
 - ssmmessages:CreateControlChannel
 - ssmmessages:CreateDataChannel
 - ssmmessages:OpenControlChannel
 - ssmmessages:OpenDataChannel
 Resource: "*"
- Effect: Allow
 Action:
 - s3:GetEncryptionConfiguration
 Resource: "*"
```

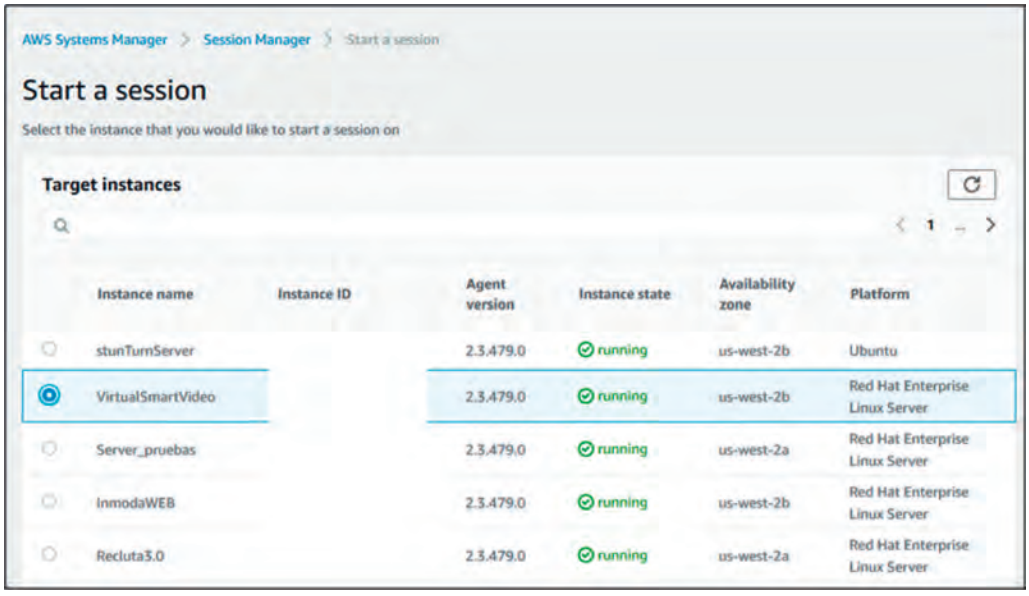


Figura 12. Iniciar sesión en EC2 con Session Manager

- *Configuración de la rutina de automatización Stop/Start en los servidores de prueba.* Para ejecutar rutinas de automatización, SSM ofrece el servicio de administrador de estados y configuración, con él se pueden controlar detalles de configuración –i.e., parámetros del servidor, definiciones de antivirus, ajustes de firewall, respaldos automáticos, etc.) y definir políticas de configuración para los servidores mediante la consola de administración de AWS o usar scripts existentes, módulos de PowerShell o compendios de Ansible directamente desde GitHub o *buckets* de Amazon S3. SSM implementa automáticamente las configuraciones en las instancias seleccionadas, en el momento y con la frecuencia que el usuario administrador defina [34].

  Para reducir costos de operación, se tomó la decisión de configurar dos rutinas de automatización: una para apagar los servidores de desarrollo y pruebas a partir de las siete de la noche, otra para encenderlos a las siete de la mañana, cada una de ellas rutinas se configuró utilizando documentos y scripts oficiales de Amazon, disponibles dentro del administrador de configuraciones del SSM. En las rutinas fueron incluidas las siete instancias del grupo de recursos ServerPruebas. La configuración, gracias al manejo del grupo de recursos, se ve simplificada.
- *Rutina de automatización del respaldo de instancias.* Se creó una rutina que permite generar respaldos de las configuraciones actuales de las instancias EC2 de la cuenta de AWS de la empresa y se configuró para que cree un AMI cada domingo. Esta rutina se ejecuta de manera simultánea en todas las instancias EC2 presentes en la cuenta de la compañía.
- *Creación de una guía de referencia.* El SSM es un servicio muy completo que permite gestionar de manera centralizada toda la infraestructura; en el modelo operativo propuesto en este proyecto, este tipo de herramientas habilita la automatización de tareas de gestión y mantenimiento de infraestructura, lo que le permite a los equipos técnicos dedicar más de su tiempo a agilizar los procesos operativos de la empresa. Con su implementación se resuelve una de sus principales necesidades.

  Se desarrolló una guía completa (disponible en [35]) que explica en detalle cada una de las implementaciones descritas. Ella puede ser consultada en caso de dudas sobre la implementación de servicios adicionales o ante la necesidad de cambios en la configuración de alguna de las opciones expuestas.

FORMALIZACIÓN DE PROCESOS DEL MODELO OPERATIVO

Mejorar la infraestructura ayuda a una operación más ágil, pero no constituye un modelo operativo que habilite el desarrollo y aprovisionamiento de servicios de TI, para lograrlo es necesario diseñar e implementar el modelo desde una visión estratégica que impacte la cultura organizacional y la operación de la compañía. Las organizaciones deben enfocarse en formalizar los procesos contemplando su mejoramiento continuo para potenciarse en todas sus áreas y traducir esa mejoría en mejores aplicaciones y servicios que cumplan los requerimientos de los clientes.

Las empresas deben apuntar al establecimiento de una estructura basada en procesos para tener claridad en su gestión interna [36]. Para lograr la visión general se procede con la elaboración de un mapa de procesos [37], la representación gráfica de la estructura de procesos que conforman el sistema de gestión de la organización. Los procesos se categorizan en: estratégicos, operativos, de apoyo y de evaluación; el mapa se construye con los flujos de información y sus relaciones [38].

La realización de un mapa de procesos detallado es un proyecto asociado con la certificación ISO 9001:2015 [39] que sobrepasa el alcance de este proyecto, por lo que fue necesario diseñar y efectuar una herramienta de diagnóstico y priorización para clasificar los procesos clave. Se diseñó una herramienta de diagnóstico de una sola vista que permite obtener la percepción de la empresa acerca de sus procesos y ver la integridad de cada grupo de procesos y evaluarlos de manera conjunta.

La herramienta es una de las principales contribuciones de este proyecto, logra integrar las opiniones y percepciones de todos los colaboradores de la empresa, lo que facilita priorizar los procesos clave. La herramienta está diseñada para PYME y está constituida por cinco grupos de procesos principales (Tabla 3), centrados en procesos tecnológicos que deben ser la base de la operación de la compañía, cada uno conformado por procesos clave de la compañía (Tabla 4). Cada grupo se descompone en sus procesos principales, una estructura jerárquica muestra diversos niveles de relevancia lo que provoca que los elementos más grandes tengan mayor prioridad que los elementos inferiores.

Para exponer la herramienta se decidió utilizar una estructura de diagrama de Venn que muestra la integración y sinergia entre conceptos (Figura 13). Los círculos de los grupos de procesos presentan colores neutros, mientras que los

Tabla 3. Herramienta de diagnóstico y priorización: grupos de procesos

Grupo	Descripción
Gestión del servicio	Para su definición se utiliza como referencia el marco de trabajo de ITIL [Information Technology Infrastructure Library], en él: los servicios constituyen el medio para entregar valor a los clientes sin tener exposición a costos y riesgos específicos; la gestión del servicio cuenta con un ciclo de vida de cinco fases – estrategia, diseño, transición, operación y mejora continua del servicio; y realizar la gestión del servicio implica, en primera medida, gestionar cada fase del ciclo de vida de manera óptima [40].
Enfoque ágil	El agilismo se fundamenta en cuatro valores y doce principios [16], los enfoques y metodologías ágiles son términos generales que cubren una gran variedad de marcos de trabajo y métodos, lo que posiciona a este movimiento como: cualquier tipo de enfoque, técnica, marco, método o práctica que cumpla con los valores y principios del Manifiesto Ágil [16]. En general, existen dos estrategias para cumplir con ellos: adoptar un enfoque ágil formal, diseñado intencionalmente y probado para lograr los resultados deseados; e implementar cambios en las prácticas de un proyecto, ajustándose al contexto del proyecto para lograr progresos en una característica central [3]. En esta segunda estrategia, se considera dividir un proyecto grande en varios incrementos, si eso funciona para el contexto específico del proyecto.
Gestión del conocimiento	Componente fundamental en la operación de una compañía y en la generación de valor, genera innovaciones en productos y procesos orientados a mejorar progresivamente los sistemas de gestión organizacional. La gestión del conocimiento puede entenderse como el proceso de generación de valor a partir de los activos intangibles de una organización [41] y definirse como el conjunto de actividades orientadas hacia la creación, la puesta en común, el intercambio y la aplicación del conocimiento fundamental para la mejora de los resultados de la organización, lo que lo convierte en un componente estratégico fundamental [42].
Gestión de la seguridad	Por ser un requerimiento no funcional de los servicios, se tiende a considerarla al final del proceso. Con ello se corre el riesgo de tener brechas que ponen en peligro la información de los clientes. Este grupo de procesos busca diagnosticar de manera superficial un Sistema de Gestión de la Seguridad de la Información [SGSI], esto es el conjunto de políticas y procedimientos para gestionar sistemáticamente los datos confidenciales de una organización. El objetivo de un SGSI es minimizar el riesgo y garantizar la continuidad del negocio al limitar de manera proactiva el impacto de una violación de seguridad. Un SGSI generalmente aborda el comportamiento y los procesos de los empleados, los datos y la tecnología, y puede dirigirse a un tipo particular de datos –como los datos de los clientes– o implementarse de una manera integral, que se convierta en parte de la cultura de la empresa [43].
Prácticas continuas	Están destinadas a ayudar a las organizaciones a acelerar su desarrollo y a entregar funciones de software sin comprometer la calidad. Además permiten: obtener comentarios rápidos del proceso de desarrollo de software y de los clientes; tener lanzamientos frecuentes y confiables, que conducen a una mayor satisfacción del cliente y a una mejor calidad del producto; y fortalecer la conexión entre los equipos de desarrollo y operaciones eliminando las tareas manuales [17].

Tabla 4. Herramienta de diagnóstico: grupos de proceso y sus procesos clave

Grupo de procesos	Proceso clave	Criterio
Gestión del servicio	Gestión del ciclo de vida del servicio	Gestionar de manera óptima las cinco fases del ciclo de vida descritas por ITIL [40]: estrategia del servicio, diseño del servicio, transición del servicio, operación del servicio y mejora continua del servicio.
	Despliegue de software y servicios	Gestionar de manera adecuada el despliegue de los servicios al cliente, entregando valor (centrado en la transición y operación del servicio).
	Gestión de requerimientos	Gestionar los requerimientos de los clientes para cada servicio de TI (centrado en la estrategia del servicio).
	Pruebas de calidad entrega del servicio	Gestionar las pruebas de calidad de los servicios para entregar al cliente valor con una solución acorde con sus necesidades.
	Soporte técnico	Gestionar los casos de soporte del servicio y desde ahí identificar oportunidades de mejora (centrado en la operación del servicio).
	Buenas prácticas en la gestión de la infraestructura	Gestionar de manera centralizada la infraestructura de la compañía en AWS (este proceso clave se enuncia en este grupo debido a su importancia dentro del modelo).
	Involucramiento del cliente	Tener una alta tasa de retroalimentación del cliente en cada iteración del desarrollo y operación del servicio es un elemento crucial.
	Priorización de características	Tener una buena retroalimentación facilita priorizar las características que más necesita el cliente para el servicio en desarrollo o para mejorar la operación del servicio actual.
Enfoque ágil	Involucramiento del equipo	Todos los miembros del equipo deben estar involucrados con el proyecto, comprender el contexto, entender al cliente y tener claras las características a desarrollar.
	Claridad en las prioridades	Las prioridades deben ser claras para todo el equipo, así se evita desperdiciar tiempo en desarrollar características que el cliente no necesita.
	Adaptación al cambio	Los equipos ágiles deben ser adaptables y entender el cambio como un componente normal de los proyectos.
	Entendimiento del estado del proyecto	Debe ser claro el punto de desarrollo del proyecto. Saber en qué estado se encuentra le ayuda a los equipos a comprender las prioridades y priorizar las tareas.
	Planeación eficiente	Planear las actividades del equipo con el fin de lograr todos los objetivos planteados en cada incremento de desarrollo del servicio.
	Estructura del proyecto	El proyecto debe contar con una estructura clara, conocida por todos los miembros del equipo, así se puede hacer un desarrollo paralelo y lograr una entrega temprana del servicio.

Tabla 4. Herramienta de diagnóstico: grupos de proceso y sus procesos clave (cont.)

Grupo de procesos	Proceso clave	Criterio
Enfoque ágil (cont.)	Equipo autogestionado	El equipo mismo es responsable de rendir cuentas en la mayoría de los aspectos del desarrollo y operación de un servicio, no existe la figura de un jefe que asigna el trabajo, sino que cada miembro tiene la madurez necesaria para trabajar en las actividades requeridas para cumplirle al cliente.
Gestión del conocimiento	Base de conocimientos	Contar con esta base permite realizar una consulta rápida sobre problemas que se presenten en los servicios, algoritmos de desarrollo, conceptos y cualquier información relevante para la operación de los servicios; al seleccionar los documentos, enlaces y libros que pueden ser utilizados en el contexto de la compañía, se reducen los tiempos de consulta.
	Claridad en los procesos	Dado que los procesos son el eje fundamental en la operación de una organización, cada miembro del equipo debe tener claro el manejo de los procesos y cómo su trabajo impacta en los resultados de la empresa.
	Procesos de capacitación	Capacitar a los colaboradores en los procesos internos de la compañía y fomentar el aprendizaje activo de diversos temas tecnológicos y gerenciales.
	Repositorio documental	Para lograr que la compañía tenga acceso rápido a los documentos de la empresa –i.e., propuestas comerciales, documentos de elicitación de requerimientos, llaves de acceso, programación de tareas, etc. –, es importante tener un repositorio documental centralizado en donde se organiza la documentación para cada proyecto.
	Repositorio de código	El equipo de ingenieros debe tener un repositorio central del código de cada uno de los servicios ofrecidos; el código de las aplicaciones y servicios debe tener un respaldo, no debe existir solamente el código desplegado en el ambiente de producción.
	Documentación del código	Además de contar con un repositorio de código, el código fuente debe estar debidamente documentado para que cada miembro del equipo de ingenieros sepa qué hace cada método y algoritmo de las aplicaciones y servicios ofrecidos por la empresa.
Gestión de seguridad	Conexión segura	Se deben establecer canales seguros de comunicación de cada uno de los servicios sin importar el origen y destino de la comunicación.
	Integridad de la información	Toda la información debe estar almacenada en un sitio seguro y estar respaldada para evitar problemas de corrupción en la información.
	Autenticación de usuarios	Cualquier tipo de usuario que desee acceder a cualquier servicio debe autenticarse y cumplir con una política de contraseñas segura.

Tabla 4. Herramienta de diagnóstico: grupos de proceso y sus procesos clave (cont.)

Grupo de procesos	Proceso clave	Criterio
Gestión de seguridad (cont.)	Control de acceso	Una vez se autentica un usuario, debe ser claro qué actividades tiene autorizadas realizar dentro del servicio o la aplicación.
	Registro de actividades	Se considera realizar un seguimiento de la actividad del usuario para tener trazabilidad completa a través de logs.
	Estrategias de prevención de ataques	Dentro del SGSI se deben implementar estrategias de prevención de ataque que ayuden a detectar y corregir los problemas durante un ataque informático.
	Licenciamiento de software	Se debe contar con software licenciado y con soporte del fabricante para evitar fallos en los servicios y brechas de seguridad.
Prácticas continuas	Integración continua	Integración del trabajo realizado por el equipo de desarrollo, de una manera automatizada.
	Entrega continua	Su objetivo es garantizar que una aplicación, al pasar con éxito las pruebas, esta lista para producción
	Despliegue continuo	Su objetivo es implementar de manera automática y constante cada cambio de la aplicación y los servicios en el entorno de producción.
	Monitoreo continuo	Se centra en validar que el desempeño de la aplicación y los servicios se sostenga. Las métricas principales son: rendimiento de la aplicación, rendimiento del sistema, comportamiento del usuario y experiencia de usuario.

círculos de los procesos clave están en blanco. Cada uno de ellos es evaluado al momento de aplicar la herramienta. Para indicar la percepción sobre su estado, individualmente se debe rellenar el círculo del proceso con un color, así: verde, si el proceso se encuentra en un estado estable y funciona de manera correcta y eficiente, si no es necesario implementar una mejora; amarillo, si el proceso se encuentra en un estado estable, pero se identifican algunas oportunidades de mejora que deben implementarse para lograr una operación eficiente; naranja, si el proceso existe pero no es estable ni eficiente y requiere de la implementación de varias oportunidades de mejora para lograr su estabilidad; y rojo, si el proceso no existe en la compañía o si está previsto su desarrollo a futuro. Para su procesamiento se relaciona cada color con un número (de 3 a 0, respectivamente) y para su consolidación se realiza un promedio simple.

Una vez validada la herramienta se procedió a su aplicación con los interesados clave de la compañía: los gerentes y los líderes de área, para obtener

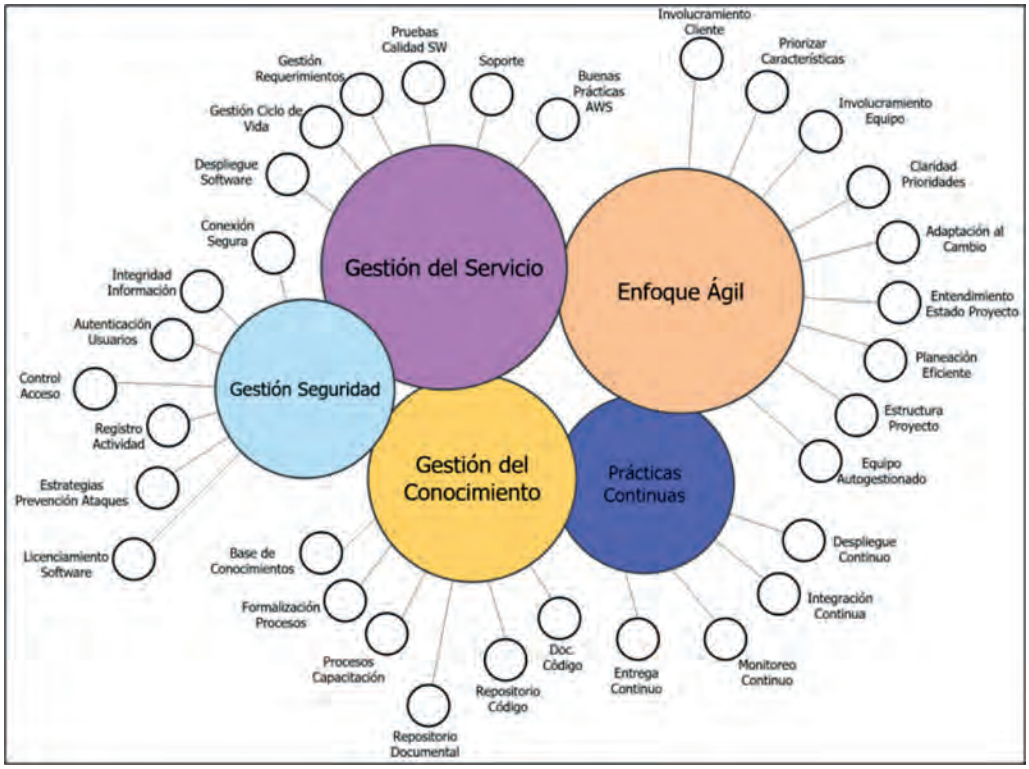


Figura 13. Herramienta de diagnóstico

su percepción respecto a los procesos claves de la compañía y así diagnosticar y priorizar los procesos que más afectan a la organización. Previo a la aplicación de la herramienta se desarrolló un taller donde: se presentó el mapa de procesos; se explicó la importancia de la gestión de procesos; se presentaron los grupos de proceso y sus procesos clave –para asegurar que el equipo comprendía cada uno y su importancia en el modelo en construcción–; y se explicó la herramienta de diagnóstico, su estructura y la razón de su implementación. Hecho esto, cada interesado clave diligenció la herramienta. Sus resultados fueron consolidados por separado en dos grupos, gerentes y líderes, con el fin de determinar la brecha entre ambos y así contrastar la visión del equipo de ingeniería (los líderes) con la visión del equipo directivo (los gerentes).

El primer grupo estuvo formado por los líderes: técnico y de web, de desarrollo móvil y de proyectos. Como se aprecia en la Figura 14, que corresponde al consolidado de sus respuestas, no existe un proceso que consideren estable

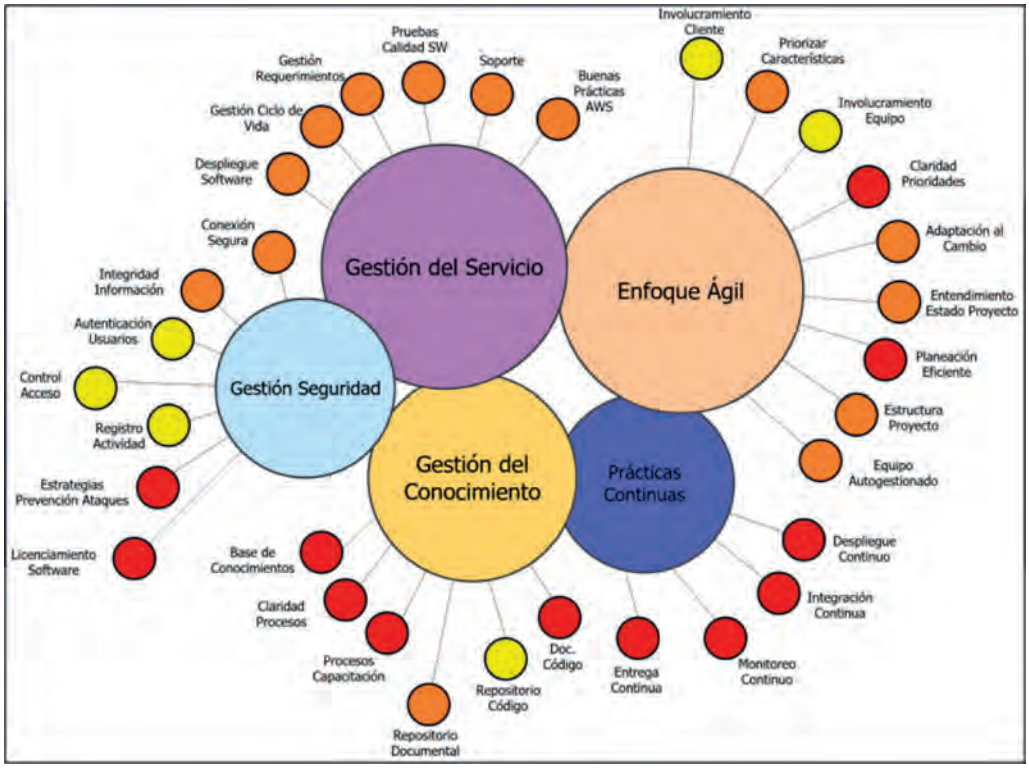


Figura 14. Consolidado de la percepción del equipo de ingeniería

y eficiente. De los 32 procesos clave, consideran seis (18.75 %) estables pero con oportunidades de mejora; catorce (43.75 %) que existen, pero que no son estables ni eficientes; y los doce restantes (37,5 %), que debería empezar su implementación. Un detalle de las respuestas de cada líder para cada proceso está disponible en [25, Tablas 12 a 16].

La Figura 15 corresponde al consolidado de las respuestas del grupo directivo, en ella se puede apreciar que: solo consideran estable y eficiente a uno de los procesos clave; valoran diez (31.25 %) como estables pero con oportunidades de mejora; otros diez, como existentes pero no estables ni eficientes; y los once restantes (34.37 %), como que se debería iniciar su implementación.

Como se puede apreciar, existe una diferencia entre la percepción de ambos grupos, por ello resulta necesario priorizar los procesos clave que se verán afectados por el modelo operativo propuesto en este proyecto. Para hacerlo se

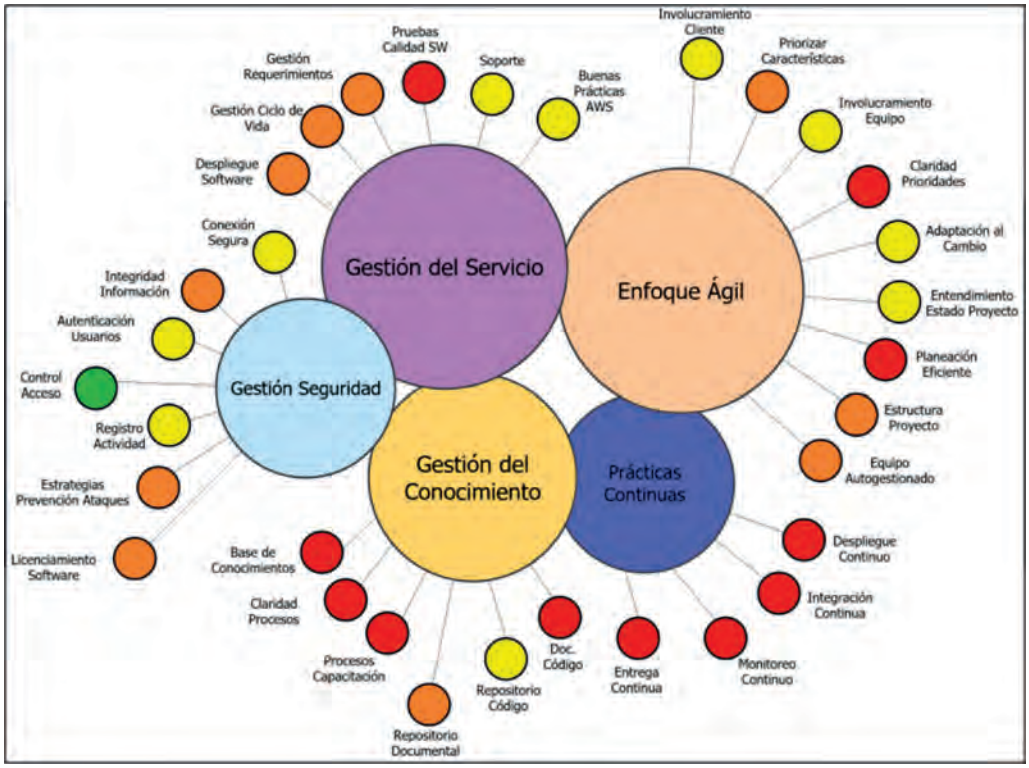


Figura 15. Consolidado de la percepción de la gerencia

utiliza el criterio de mayor brecha entre los resultados de cada proceso clave, utilizando el valor absoluto de la diferencia entre ambos: si la diferencia es mayor a dos, el proceso debe ser priorizado; si en ambos casos, el proceso fue percibido como no existente o próximo a desarrollar, también se prioriza. Se decide utilizar esta estrategia de priorización porque a través de su aplicación es posible mejorar los procesos que para las mismas personas de la compañía están en un estado negativo, tienen una mala percepción y requieren una pronta mejora. La Tabla 5 presenta los resultados de este ejercicio (para un mayor detalle favor referirse a [25, Tablas 17-21].

Los resultados del ejercicio anterior se presentaron a la gerencia con el fin de discutir el alcance del producto final del proyecto, los resultados los sorprendieron pues consideraban que tanto los procesos como el soporte eran estables y eficientes para todos en la organización; fue un ejercicio enriquecedor debido a que los gerentes tuvieron claro que con las mejoras

Tabla 5. Procesos clave priorizados

	Proceso
Enfoque ágil	Priorizar características
	Claridad en las prioridades
	Planeación eficiente
Prácticas continuas	Despliegue continuo
	Integración continua
	Pruebas continuas
	Monitoreo continuo
Gestión del conocimiento	Base de conocimientos
	Formalización procesos
	Procesos de capacitación
Gestión del servicio	Soporte

en la infraestructura no era suficiente y que aún existen muchos procesos y componentes de la organización susceptibles de mejora.

Como resultado de esta discusión, se definieron los procesos que serían abordados por el modelo operativo desarrollado en este proyecto y los que serían responsabilidad de mejora completa por parte de la empresa. El alcance final del proyecto contempló los siguientes procesos clave: priorizar características, claridad de las prioridades, planeación eficiente, integración continua, entrega continua y formalización de los procesos de desarrollo de requerimientos.

Antes de continuar, es importante mencionar que luego de la presentación de los resultados obtenidos con la herramienta de diagnóstico, el equipo de desarrollo decidió implementar repositorios de código versionables utilizando la herramienta Git, con el fin de tener un respaldo del código fuente de las aplicaciones y servicios desarrollados.

FORMALIZACIÓN DE LOS PROCESOS DE DESARROLLO DE REQUERIMIENTOS

Antes de avanzar con los grupos de procesos del enfoque ágil y las practicas continuas, el desarrollo del modelo operativo debe iniciar con los procesos de desarrollo de requerimientos. La empresa maneja el desarrollo de aplicaciones y servicios como requerimientos de los clientes, para ello cuenta con un proceso establecido por el equipo de ingenieros, pero carente de

formalización. A continuación se formaliza cada uno de esos procesos y se expone una propuesta de mejora. Los procesos involucrados en el desarrollo de un nuevo requerimiento son, en orden: nuevo requerimiento, asignación de requerimiento, desarrollo de un nuevo requerimiento, asignación de una No Conformidad [NC] y solución de una NC.

Nuevo requerimiento

Como se ha indicado, las necesidades de los clientes se formalizan como requerimientos, estos son la base del desarrollo y operación de cada servicio y aplicación; cuando se planea el desarrollo y aprovisionamiento de un nuevo servicio, también se planifica según sus requerimientos. Un nuevo requerimiento puede llegar como un documento formal del cliente o como una necesidad expresada por él.

Si el requerimiento es claro para el líder de proyecto y los gerentes, se procede a estimar las horas de desarrollo necesarias para resolverlo y a evaluar su complejidad entre los líderes web, móvil y de proyectos. Posteriormente se valida si el requerimiento se encuentra dentro de una garantía de servicio, de ser así, se procede a establecer una fecha tentativa de entrega y se asigna el requerimiento a un ingeniero para su desarrollo, en caso contrario, el requerimiento es cotizado y negociado con el cliente con una fecha tentativa de entrega; cuando el cliente acepta las condiciones de desarrollo y envía la orden de compra, el requerimiento es asignado.

Una vez asignado el requerimiento el ingeniero avanza con el desarrollo y los gerentes y líderes se encargan de la revisión del avance y el cumplimiento. Cuando el desarrollador termina el requerimiento, realiza unas pruebas: si no son exitosas, regresa a la fase de desarrollo, si lo son, pasan a pruebas de usuario, las cuales son generalmente realizadas por la líder de proyectos, quien hace las últimas validaciones. Si el resultado no es satisfactorio, el requerimiento se devuelve al desarrollador para su corrección y el proceso se reinicia. Cuando las pruebas de usuario culminan, el requerimiento se entrega al cliente, quien realiza sus pruebas; si encuentra fallos, genera una NC, que debe ser corregida por el equipo. En la Figura 16 se presenta en detalle el flujo de completo de este proceso.

El proceso actual tiene dos inconvenientes: el primero, que al estar basado en un modelo de ciclo de vida en cascada, es muy secuencial y solo le entrega el resultado al cliente al terminar su desarrollo, lo que en la práctica implica

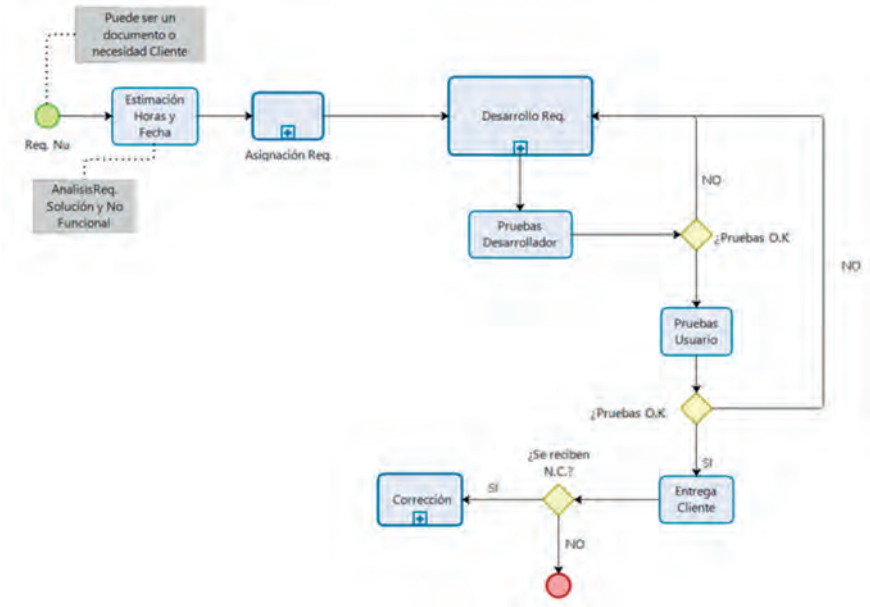


Figura 16. Proceso "Nuevo requerimiento"

un bajo involucramiento del cliente y por ello un mayor riesgo de generar una no conformidad generada porque los requerimientos no son elicitados de la manera adecuada; el segundo, que las pruebas que se hacen son solo de usabilidad, no se validan aspectos también importantes, como son: el rendimiento, la seguridad y la conectividad, entre otros.

Asignación de un requerimiento

Una vez que se acepta el desarrollo de un nuevo requerimiento, se le asigna a un ingeniero de desarrollo. El proceso (Figura 17) inicia evaluando la prioridad de dicho requerimiento para la gerencia: si es alta, se le asigna inmediatamente a alguno de los líderes para que inicie lo antes posible su desarrollo; si no, el requerimiento es evaluado entre la gerencia y los líderes para su asignación: si es categorizado como simple, se le asigna a un desarrollador; en caso contrario, pasa por una etapa de evaluación de la tecnología y el lenguaje de programación para ser asignado al ingeniero con mayor dominio del tema. Adicionalmente, si el requerimiento tiene un grado de complejidad alto o es de una tecnología emergente y requiere un trabajo de experimentación, se evalúa el ingeniero con mayor experiencia para asignarle el requerimiento.

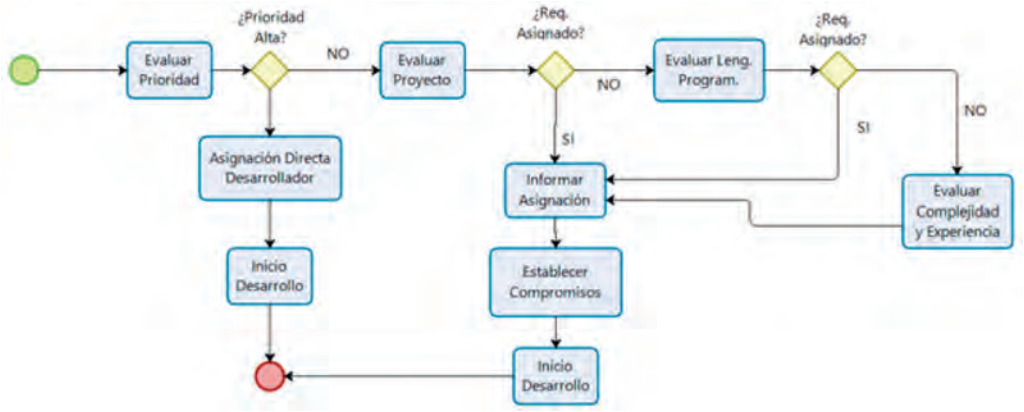


Figura 17. Proceso "Asignación de un requerimiento"

El proceso actual tiene varios inconvenientes: sobrecarga a los ingenieros más experimentados, quienes reciben una mayor cantidad de requerimientos; entrega un requerimiento completo a una sola persona para su desarrollo, sin dividirlo en tareas que puedan ser realizadas en conjunto; y, como la mayoría de los requerimientos son de prioridad alta, no se realiza una evaluación detallada de la asignación.

Desarrollo de un requerimiento

Cuando a un ingeniero de desarrollo se le asigna un nuevo requerimiento, inicia el proceso de desarrollo con análisis de requerimientos, sin embargo, en la mayoría de las ocasiones, dicho análisis no se realiza, sino que se pasa de inmediato a la creación de una rama en el repositorio. El ingeniero hace una copia de seguridad en su repositorio local, crea la nueva rama e inicia el desarrollo del requerimiento; cuando termina, ejecuta unas pruebas locales, si ellas fallan, regresa al desarrollo del requerimiento, si no, se lleva el desarrollo del requerimiento al repositorio remoto, utilizando el comando git push.

Una vez se encuentra el código fuente en el repositorio central, se realizan unas pruebas adicionales del desarrollador en los servidores de prueba: si todo sale bien, se reporta la finalización del desarrollo del requerimiento y se entrega a la líder del proyecto; en caso de fallas, se regresa a la fase de desarrollo para hacer las correcciones. Luego, la líder de proyecto realiza unas pruebas de usuario finales para validar el funcionamiento del requerimiento: si no identifica nada malo, entrega el requerimiento resuelto al cliente con los

resultados de las pruebas. En este punto el cliente realiza una evaluación del requerimiento y genera NC para las fallas que encuentre. Esta secuencia se presenta en la Figura 18.

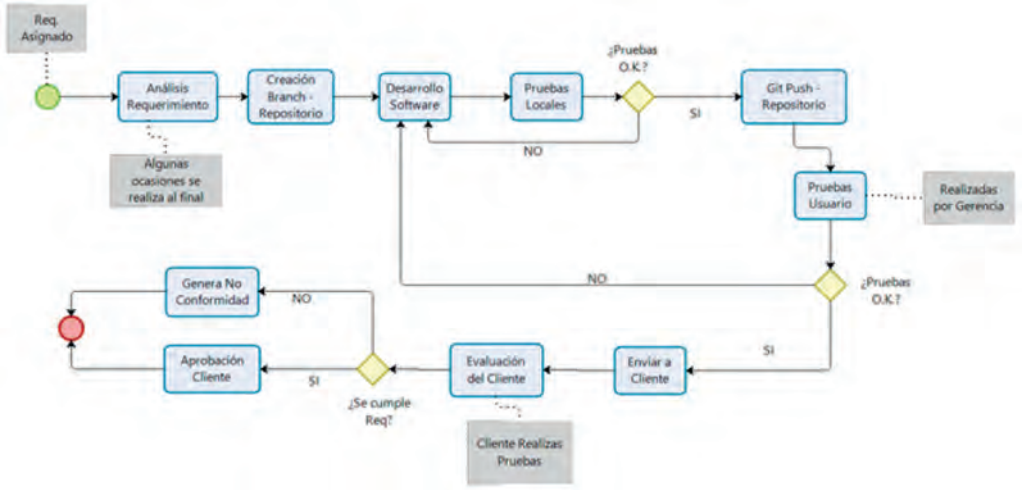


Figura 18. Proceso "Desarrollo de un requerimiento"

El desarrollo de un requerimiento, al ser un proceso monolítico, que solo le entrega el resultado al cliente al finalizar su desarrollo, implica falta de comunicación con él, lo que da como resultado poca claridad en las restricciones y especificaciones; además, la falta de análisis inicial del requerimiento por parte del desarrollador, los líderes y la gerencia, hace que se entregue al cliente algo que no necesita, lo que disminuye su satisfacción y afecta a la empresa.

Asignación de una no conformidad

Como se evidencia en los procesos descritos, la falta de comunicación constante con el cliente provoca fallas en la entrega del requerimiento final y ello la generación de NC, las cuales, según la ISO 9000:2005, constituyen un incumplimiento de un requisito del sistema, sea este especificado o no [44]. Como se aprecia en el flujo de la Figura 19, cuando se recibe una NC, ella es revisada por la líder del proyecto y la gerencia, quienes evalúa la prioridad para su solución y los requerimientos: si la prioridad es alta, la NC es inmediatamente asignada a un desarrollador; si no, con los líderes web y móvil

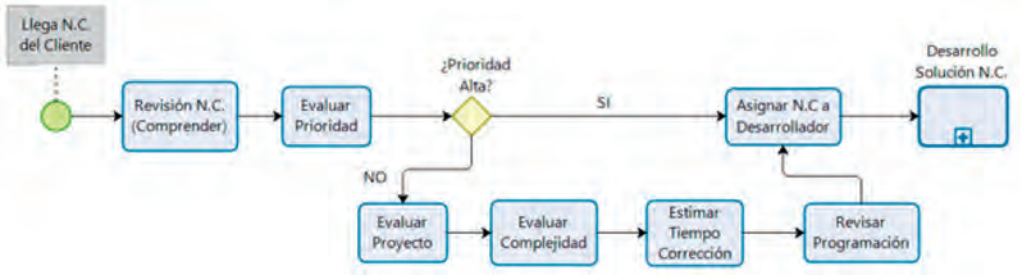


Figura 19. Proceso "Asignación de una no conformidad"

se evalúa el proyecto, la complejidad y el tiempo estimado de corrección, se revisa la programación del equipo y se asigna la corrección al primer ingeniero disponible que pueda solucionarla.

Desarrollo de la solución de una no conformidad

La líder de proyectos, al asignar una NC, se la explica al ingeniero a cargo de su solución. Él la desarrolla con la constante revisión y seguimiento de parte de la gerencia y de la líder de proyectos. Aquí el proceso es igual al del desarrollo de un nuevo requerimiento, con pruebas locales, actualización al repositorio remoto, pruebas de desarrollador y, finalmente, pruebas de usuario, si las pruebas de usuario final resultan exitosas, se informa al cliente, quien revisa la solución: si el cliente considera resuelta la NC, ella se cierra; de lo contrario, se reabre y el proceso reinicia. En la Figura 20 se describe este proceso.

En este punto, los principios ágiles cobran un sentido fundamental en el desarrollo de este proyecto, debido a que la poca interacción con el cliente; el desarrollo monolítico, con un ciclo de vida en cascada; la poca claridad en las tareas; y el mal manejo de prioridades, entre otros factores, provoca que los procesos de desarrollo de servicios y aplicaciones sean considerados por el cliente como lentos y de baja calidad, lo que afecta su propuesta comercial y su valor competitivo.

DISEÑO DEL MODELO OPERATIVO ÁGIL

El diseño del modelo operativo toma como referencia los principios de DevOps [1], [16] y [45]. Deben mejorarse: los procesos de formalización de requerimientos, la colaboración y comunicación del equipo, la asignación

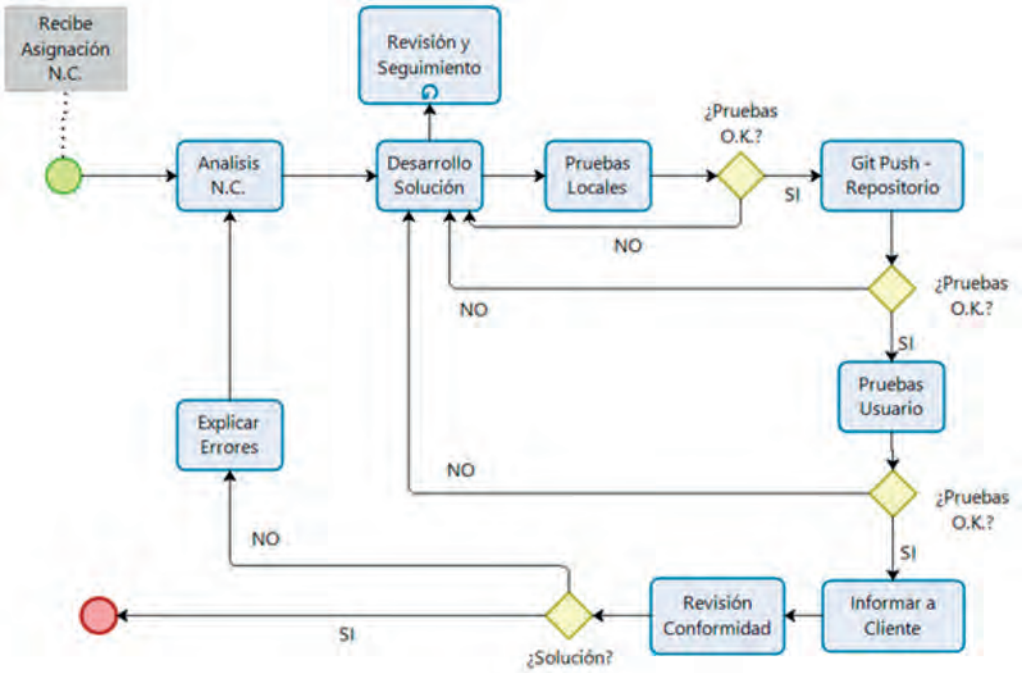


Figura 20. Proceso "Desarrollo de la solución de una no conformidad"

de tareas, el desarrollo de soluciones en paralelo, las practicas continuas, el involucramiento del cliente, el despliegue automático de infraestructura, la planeación de tareas y la priorización actividades; cada uno de los elementos hace parte fundamental del modelo operativo propuesto en este proyecto. A continuación se expone el proceso realizado para lograr un modelo operativo que permita el despliegue ágil y flexible de servicios de TI. El Modelo contempla cuatro etapas principales: requerimientos y tareas, tablero de priorización, *pipeline* de infraestructura y *pipeline* de desarrollo. Cada uno de estos componentes constituye el modelo planteado, el cual debe verse como un todo.

Roles, requerimientos y tareas

La empresa cuenta con un área de desarrollo general, unos líderes de desarrollo web y móvil y una líder de proyecto, esta estructura jerárquica presenta roles que dependen del tipo de tecnología y del ambiente dónde será desplegado un servicio (web o móvil) y no hay responsables claros del manejo de la

infraestructura, lo que provoca dificultades en la organización del equipo y en la planeación de actividades. Se propone crear una estructura organizacional basada en los siguientes equipos:

- *Chief executive officers*. Equipo de gerencia, responsable de la propuesta comercial, la presentación del portafolio de servicios y del primer contacto con el cliente para tomar la oportunidad y volverla un proyecto; debe estar al tanto de toda la actividad de la compañía y de las áreas que no son de intereses para el desarrollo de este proyecto, tales como contabilidad y recursos humanos.
- *Project team*. En lugar de una persona a cargo un proyecto, un equipo que cuente con un líder, un gerente y un área de diseño. Este grupo coordinará todos los proyectos y velará por el cumplimiento de los compromisos adquiridos con el cliente.
- *Infrastructure team*. Grupo responsable de desplegar, aprovisionar y gestionar la infraestructura en AWS para la operación de los servicios, estará conformado también por personal de soporte, con el fin de agilizar la resolución de problemas.
- *Developer team*. Equipo encargado de desarrollar cada aplicación y servicio requerido por el cliente, debe reunir conocimientos de diferentes lenguajes de programación, tecnologías y arquitecturas, para así poder atender cualquier oportunidad comercial.
- *Quality team*. Grupo encargado de velar que todo producto, aplicación, servicio o incremento que se entregue al cliente cumpla con las validaciones planteadas en los requerimientos, y así velar por entregar valor al cliente presentando soluciones que se ajustan a sus requerimientos y necesidades.

El proceso de mejoramiento inicia con una propuesta de cambio en el elemento fundamental para el diseño, desarrollo, verificación y puesta en funcionamiento de una aplicación o servicio: los requerimientos. Tal como se expresó, no se realiza un análisis de requerimientos formal y no se sigue una metodología de elicitación, lo que provoca: falta de claridad en las necesidades del cliente, desarrollo de soluciones que no agregan valor al cliente y una gran cantidad de NC. Para dar solución a esta problemática, se propone el manejo de requerimientos como historias de usuario, un mecanismo de manejo de requerimientos ágil, debido al alto involucramiento del cliente y a una estructura flexible y adaptativa.

Los requisitos ágiles manejan tres niveles para la gestión de proyectos: en el primero, visión, se producen los temas, una colección de *epics* e historias de usuario que relacionados describen un sistema o subsistema en su totalidad; en el segundo, pila de producto, se encuentran los *epics* y las historias de usuario, las que se distinguen unas de otras por su tamaño –los *epics* son superhistorias de usuario de gran tamaño– y su granularidad, alta y baja, respectivamente; y en el tercero, pila de incrementos, se encuentran las tareas, el resultado de la descomposición de las historias de usuario en unidades de trabajo adecuadas para gestionar y seguir el avance de su ejecución [46].

El proyecto se enfoca en desarrollar las aplicaciones y servicios siguiendo el modelo de historias de usuario y tareas. Las historias de usuario son utilizadas en los métodos ágiles para la especificación de requisitos, son una descripción breve de una funcionalidad software tal y como la percibe el usuario y se componen de tres aspectos importantes [47]: una descripción escrita de la historia, utilizada para la planificación y como recordatorio; conversaciones sobre la historia, que sirven para desarrollar los detalles de la historia; y pruebas que transmiten y documentan detalles, útiles para determinar cuándo se completa una historia.

Las historias de usuario describen lo que el cliente o el usuario quiere que se implemente y se escriben con una o dos frases utilizando lenguaje común; se basan en el hecho de que el pensamiento de las personas se estructura siguiendo una narrativa, una historia, ellas desde pequeñas se capacitan para comprender personajes, deseos y motivaciones. Son, por lo tanto, la forma más fácil de adquirir y retener conocimiento [46].

Cada historia de usuario debe ser fácil de memorizar y escribir sobre una tarjeta o *post-it*. Poco antes de ser implementadas, van acompañadas de conversaciones con los usuarios y la definición de los criterios de validación asociados. Al ser consideradas historias, los cambios son bien recibidos, por lo que no vale la pena profundizar en el inicio, ya que en el momento de la implementación pueden cambiar. Los criterios de validación le permiten: al propietario del producto o usuario de negocio, confirmar que el equipo ha recogido correctamente los requisitos; y al equipo, realizar las pruebas adecuadas, desarrollar guiados por pruebas con *Test-Driven Development* [TDD] y comprobar que la historia se ha completado [46]. TDD es una práctica de programación que consiste en escribir primero las pruebas (generalmente unitarias), después escribir el código fuente que pase la prueba satisfactoriamente y luego refactorizar el código escrito [48].

Se plantea utilizar las historias de usuario por proyecto, cada proyecto tendrá un color asignado y las historias de usuario estarán asociadas a él; de cada historia de usuario –que representa un requerimiento del cliente–, se desprenderán las tareas, y la agrupación de tareas clave será considerada un incremento. Con este diseño se cumple el esquema de requisitos ágiles. Adicionalmente, se debe tener un ingeniero de calidad que realice las pruebas que cumplan con todas las validaciones planteadas por el cliente.

En el modelo operativo actual de la empresa, a cada ingeniero se le asigna un requerimiento completo para su desarrollo. Un requerimiento implica el desarrollo de varios métodos, algoritmos, clases y componentes visuales, entre otros. Desarrollar un requerimiento completo es una tarea compleja que puede tomar varios días e incluso semanas, requiere además de una manipulación del *back-end* y el *front-end* sin la supervisión de un líder o encargado del código, lo que puede provocar una mala integración en el código fuente, que se verá reflejado como mal funcionamiento en la aplicación o servicio.

Aunque cambie la forma de manejar los requerimientos, no es viable continuar con la asignación de trabajo a los ingenieros por historia de usuario completa, es necesario desglosarlas como tareas, esto es actividades atómicas que pueden ser realizadas y validarse de manera única y debe llevarse a cabo dentro de un período de tiempo definido o en una fecha determinada [49]. La acumulación del total de las tareas completa una historia de usuario.

Cabe destacar que las tareas son todas las actividades necesarias para lograr entregar la historia de usuario, por lo tanto, debe haber tareas para los equipos de infraestructura, desarrollo y calidad. Las tareas pueden ser secuenciales –aquellas que dependen de otra para poder iniciar– o paralelas –las que se pueden realizar al mismo tiempo, sin problemas de compatibilidad–. Es importante entonces, en la planeación del proyecto y el desarrollo de la historia de usuario, determinar cuáles tareas serán secuenciales y cuáles paralelas. Cada tarea debe contar con un título, el ID de la historia a la que pertenece, la actividad a desarrollar y las dependencias.

Uno de los mayores beneficios de utilizar una programación basada en tareas es la posibilidad de realizar un desarrollo paralelo de la historia de usuario y del proyecto, ya que todo el equipo de desarrollo e infraestructura se concentra en trabajar en pequeñas tareas que se van integrando, y poco a poco van construyendo la solución completa. Las tareas y las historias de usuario representan la base para mejorar los procesos de priorización de actividades,

claridad en las prioridades y planeación eficiente. Lograr el trabajo paralelo será posible con los procesos de integración y entrega continua.

Tablero de priorización

Los procesos que mayor malestar generaban eran la priorización de actividades, la claridad en las prioridades y la planeación eficiente, aspectos que se abordan con la segunda fase del modelo operativo ágil: el tablero de priorización. Para desarrollarlo, se consideraron las técnicas de priorización de actividades tomando como base el trabajo de McLLree [50], quien revisó veinte técnicas de priorización. El documento contempla un mapa, en forma de una tabla periódica (Figura 21) que ayuda a comprender lo que cada técnica tiene para ofrecer; una descripción general de cada método, con gráficos y enlaces a recursos más detallados; cinco puntos en común; y conclusiones de todos

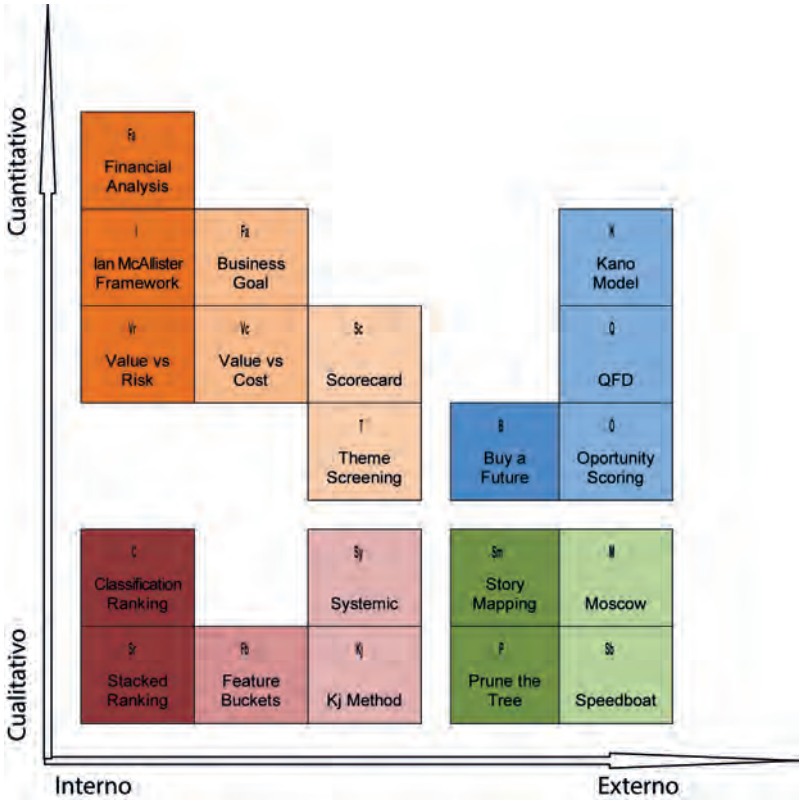


Figura 21. Técnicas de priorización [50]

estos métodos. Luego revisar las veinte técnicas, se determinó que "mapeo de historias" y "valor vs riesgo" son las más apropiadas para este caso, porque expresan la prioridad en un lenguaje visual de una sola vista, fácil de comprender.

Los mapas de historias fueron introducidos en 2005 al concluir que el manejo de historias de usuario en una lista no es la mejor forma de organizar y priorizar el trabajo que se debe realizar, sino que es necesaria una estructura más completa [50]. La organización de un mapa de historias incluye dos ejes, uno horizontal, que representa la secuencia de uso, otro vertical, que representa la criticidad: en el primero, las historias se colocan a lo largo del eje, en la misma secuencia en que las realiza el usuario; en el segundo, las historias se organizan verticalmente, en función de su importancia para el usuario, de arriba hacia abajo. Los grupos de historias relacionadas se agrupan como actividades y se dividen con una línea vertical, que agrupa y diferencia unas historias de usuario de otras. Las actividades se ubican por encima del eje vertical y no tienen ninguna secuencia de uso, simplemente son. Estas actividades componen los principales atributos del producto y no se pueden priorizar. Manejar esta estructura para organizar y priorizar el trabajo tiene al menos dos ventajas: al ser una herramienta visual, le permite a los clientes, partes interesadas y miembros del equipo de desarrollo compartir una comprensión común de qué hace el sistema; y define muy claramente cómo lanzar, de forma incremental, las iteraciones de productos que entregan porciones de trabajo completas con una sofisticación creciente [50].

Comparar el valor de lo que se debe hacer con alguna otra medida de compensación, por lo general el costo, es una forma tradicional de priorización. Cohn [51] propone considerar el riesgo como el factor de priorización y clasifica las características en dos dimensiones: valor y riesgo. El autor plantea una lucha constante entre alto riesgo y alto valor: si se evita lo riesgoso y se busca un valor alto, se puede desarrollar una gran parte del producto, hasta llegar a un obstáculo importante; pero si en cambio se hace énfasis en trabajar primero en lo de alto riesgo, se podría terminar haciendo un trabajo innecesario en funciones menos valiosas. Para priorizar se debe analizar cada riesgo y cada valor generado por separado, asignando a cada elemento un valor entero de 1 (valor/riesgo Bajo), 3 (valor/riesgo intermedio) y 5 (valor/riesgo alto). El objetivo es buscar un enfoque equilibrado, yendo primero a alto riesgo / alto valor, luego a bajo riesgo / alto valor, y finalmente a bajo riesgo / bajo valor; los artículos de alto riesgo y bajo valor se evitan (Figura 22).

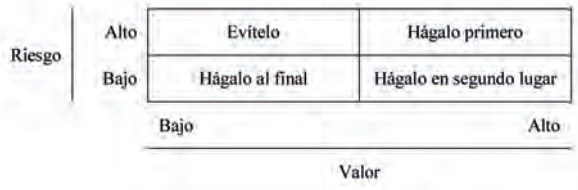


Figura 22. Valor vs riesgo

La propuesta de este proyecto contempla una combinación de ambas estrategias, maneja los ejes de tiempo y necesidad del mapa de historias y la matriz del valor vs riesgo, para priorizar las tareas: en el eje horizontal se ubica el tiempo de entrega estimado de las tareas, segmentado por semanas, con un máximo de cuatro semanas (un mes), es decir, si la tarea tiene fecha de entrega dentro de la misma semana se ubica más a la derecha y así sucesivamente; en el eje vertical se establece la criticidad de las tareas según dos criterios: las características o historias más importantes para el cliente y las tareas con mayores dependencias, es decir, las que inician una secuencia, como por ejemplo la creación de la base de datos.

Internamente en el tablero se ubican cuatro cuadrantes priorizando las tareas con menor tiempo de entrega y mayor prioridad para los clientes; en segundo lugar van las tareas con menor tiempo de entrega y menor importancia; luego, aquellas con un tiempo de entrega mayor a dos semanas y mayor importancia y, por último, las tareas con mayor tiempo de entrega y de menor relevancia para el cliente. El tablero de priorización se convierte en la interfaz principal de trabajo de la compañía y todas las tareas que se deben realizar deben quedar priorizadas en él. Así pues, el equipo de proyecto desarrolla la historia de usuario y establece una secuencia de tareas que deben ser expuestas y priorizadas en el tablero para que los equipos de desarrollo, infraestructura y calidad inicien labores.

Un cambio importante en la forma de asignación de trabajo es que el equipo de proyectos no asigna requerimientos completos a los ingenieros; con el tablero, el equipo de proyectos puede planear las actividades de la semana, ubicando las tareas debidamente diligenciadas en el tablero, según su prioridad, pero luego, cada ingeniero de los diferentes equipos consulta el tablero y selecciona la tarea que va a empezar a trabajar. Por consiguiente, el trabajo del equipo de proyecto es: velar porque las tareas se cumplan según su priorización, ver

que no se presenten tareas congeladas –que duran más de una semana en uno de los cuadrantes–; disminuir los retrasos de las tareas; involucrar al cliente en los avances; obtener retroalimentación temprana y actualizar semanalmente el tablero. Una vez los equipos de desarrollo o infraestructura toman una tarea del tablero de priorización, ella entra al pipeline de desarrollo o infraestructura, según corresponda. La Figura 23 corresponde al tablero propuesto.

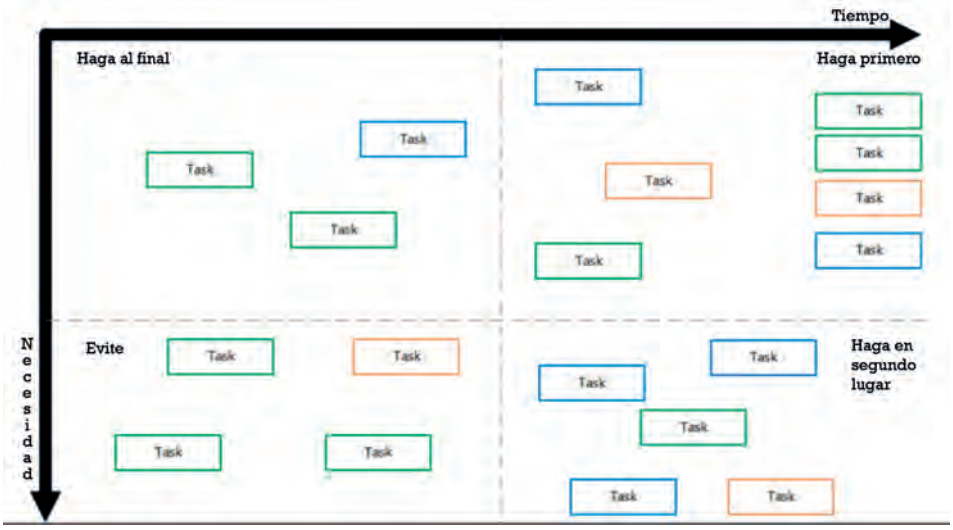


Figura 23. Propuesta de tablero de priorización

Pipeline de infraestructura

En este proyecto se entiende la infraestructura como el despliegue y la configuración de diversos recursos de AWS relacionados entre sí que soportan la operación de aplicaciones y servicios de una organización. La infraestructura de AWS contempla los recursos: VPC, Internet *gateway*, subredes, grupos de seguridad, bases de datos e instancias EC2, entre otros, y la relación entre ellos. Para automatizar la labor de desplegar, configurar y aprovisionar una infraestructura se emplea un paradigma IaC, que es parte fundamental del principio de automatización del proceso de producción de software propuesto en DevOps. IaC es un enfoque para la automatización de la infraestructura basado en prácticas de desarrollo de software, que hace énfasis en rutinas

consistentes y repetibles para el despliegue, aprovisionamiento y configuración de hardware [52]; dicha automatización se logra a través de la construcción de plantillas de programación que indican a las plataformas en la nube las instrucciones que deben ser ejecutadas. Terraform y CloudFormation son las herramientas líderes del mercado para el desarrollo de IaC.

El pipeline se encuentra fuertemente basado en el despliegue automatizado de infraestructura con IaC. Inicia con un diseño previo, elaborado por el equipo de infraestructura de la empresa, quien revisa un conjunto de plantillas IaC: si existe una para el diseño propuesto, se puede implementar; de lo contrario se debe construir una nueva utilizando CloudFormation.

Una vez la plantilla se encuentra correctamente programada, se definen unos parámetros de configuración y se procede a su ejecución; durante ella AWS CloudFormation inicia automáticamente la implementación, despliegue, aprovisionamiento y configuración de la infraestructura. Durante cada fase del proceso, AWS revisa cada recurso: en caso de error detiene todo y ejecuta una tarea de retroceso para dejar las cosas "como estaban" y notifica al ingeniero de infraestructura y al usuario que ejecuta la plantilla; si no se presenta ningún problema, el ingeniero encargado realiza una última revisión y entrega la información de la infraestructura al equipo de desarrollo y calidad para que pueda iniciar sus tareas. El diagrama del *pipeline* se presenta en la Figura 24.

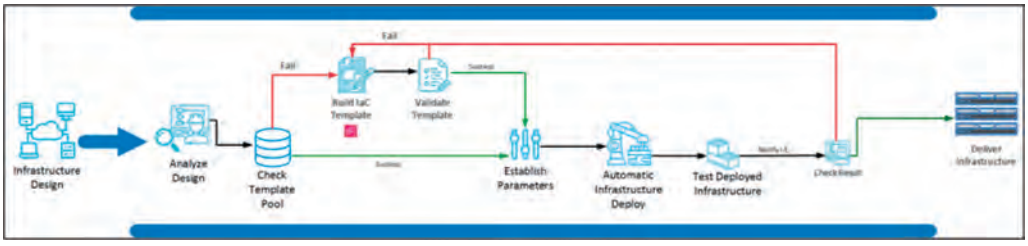


Figura 24. Pipeline de infraestructura

Pipeline de desarrollo

Este *pipeline* describe cómo va a ser construida y resuelta cada una de las historias de usuario y cuál será el modelo de desarrollo que seguirán los ingenieros de este equipo para completar cada una de las tareas; a ello se agregan las practicas de integración y entrega continua descritas en la Tabla 4.

El código fuente asociado a los proyectos, historias de usuario y tareas debe ser guardado –centralizado y versionado–, en un repositorio. Estos repositorios son manejados por cada proyecto y se dividen en ramas, cada una con un propósito específico. La rama por defecto es la rama máster, ella tiene el código en producción –es decir, la aplicación y servicio que está utilizando el usuario final–, y de ella se desprenden dos ramas principales: desarrollo y pruebas. Estos repositorios y el versionamiento se gestionan con la herramienta Git [53].

En la rama de desarrollo se maneja el código de la nueva versión que será entregada a los usuarios. Está rama cuenta con todas las características y funcionalidades nuevas de la aplicación o servicio que darán valor al cliente y tiene varias ramas "hijas", cada una representa el desarrollo de una historia de usuario específica, de tal manera que es posible desarrollar historias de usuario de manera paralela, habilitando el trabajo en equipo. La rama de pruebas presenta la misma estructura de la rama de desarrollo: una rama central, que contiene todo el código de las pruebas que serán realizadas a la versión en desarrollo antes de su paso a producción; y una rama por cada historia de usuario. Estas ramas son muy importantes ya que automatizan el proceso de validación de la historia y con ello el deseo del usuario final. Para pasar a producción, el ingeniero líder de desarrollo valida el código y realiza el paso a producción mezclando la rama de desarrollo con la rama máster. En la Figura 25 se presenta la estructura de los repositorios descrita.

El *pipeline* inicia cuando un ingeniero de desarrollo toma una tarea del tablero de priorización para iniciar su desarrollo. En este momento, el ingeniero queda bloqueado hasta que termine dicha tarea, es decir, no puede realizar una tarea adicional. Tomar las tareas nuevas y ubicarlas en el tablero de priorización es trabajo del equipo de proyectos.

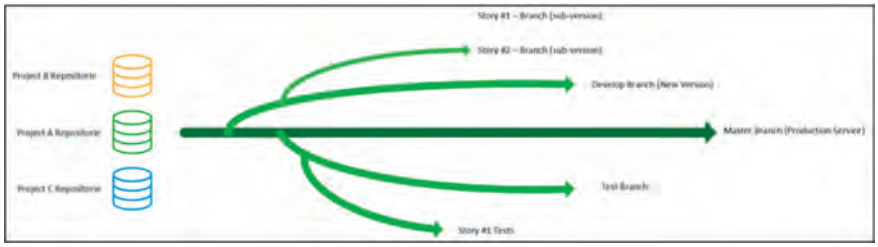


Figura 25. Estructura de los repositorios de código fuente

Para iniciar el desarrollo de la tarea, el ingeniero descarga localmente la última versión del código de la rama desde el repositorio. La tarea tiene indicado el ID de la historia, lo que debe hacer la tarea y sus dependencias. Si el ingeniero tiene dudas, puede pasar a revisar la historia de usuario con todos los detalles y entender el contexto de la tarea. Debido a que se propone seguir un modelo de desarrollo TDD, el ingeniero debe primero desarrollar las pruebas unitarias para validar cada tarea.

Cuando termina el desarrollo de las pruebas, se procede con el desarrollo de la tarea. Una vez culmina el desarrollo de la tarea, se ejecutan las pruebas unitarias definidas previamente: si estas fallan, el ingeniero regresa al desarrollo de la tarea; si todo sale bien, la tarea es comentada y se envía [push] a la rama de la historia para cargar el nuevo código dentro de la historia y completar la tarea. En este punto, el desarrollador queda liberado y puede volver al tablero de priorización y seleccionar una nueva tarea. Este proceso se puede apreciar en la Figura 26.

Al cargar el desarrollo de la tarea a la rama de la historia de usuario, se inicia un proceso de integración de la tarea con el resto de tareas desarrolladas

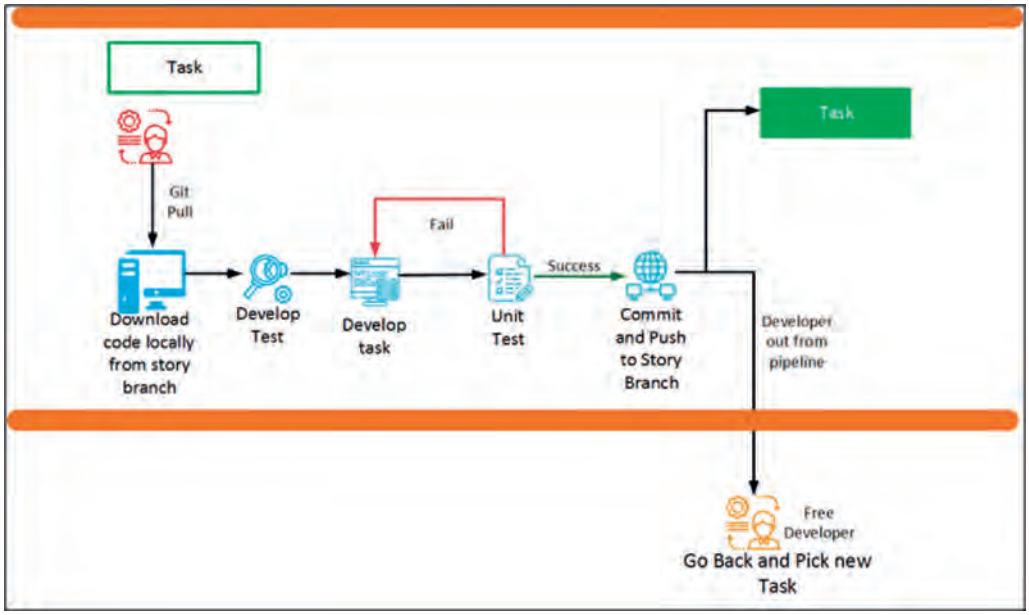


Figura 26. Inicio del pipeline de desarrollo

por el equipo para dicha historia de usuario, con ello se aplica la práctica de integración continua. Para esto, se debe contar con un servidor de CI que identifica la presencia de un cambio en la rama de una historia e inicia automáticamente un proceso de pruebas de integración; para su realización, este servidor construye un artefacto basado en el código de la historia y realiza unas pruebas de integración.

Las pruebas unitarias que hace cada desarrollador validan solamente la tarea que se está desarrollando, pero las pruebas de integración realizan pruebas de todo el código en conjunto a través del artefacto. Estas pruebas deben ser definidas por el equipo de calidad en la rama de pruebas de la historia. Una vez el servidor de CI construye el artefacto y este pasa las pruebas de integración, la rama de la historia se mezcla con la rama de desarrollo; en caso de no pasar las pruebas, el artefacto es destruido y se notifica al equipo de calidad, quien revisa lo ocurrido y programa una tarea en el tablero de priorización. Esta fase del pipeline se puede apreciar en la Figura 27.

Una vez integrada la nueva versión de la historia de usuario en la rama de desarrollo, se realiza una nueva prueba de integración sobre la rama utilizando un servidor CI diferente. Esta prueba tiene la misma estructura, se crea un artefacto y sobre él se ejecutan todas las pruebas definidas en la rama principal

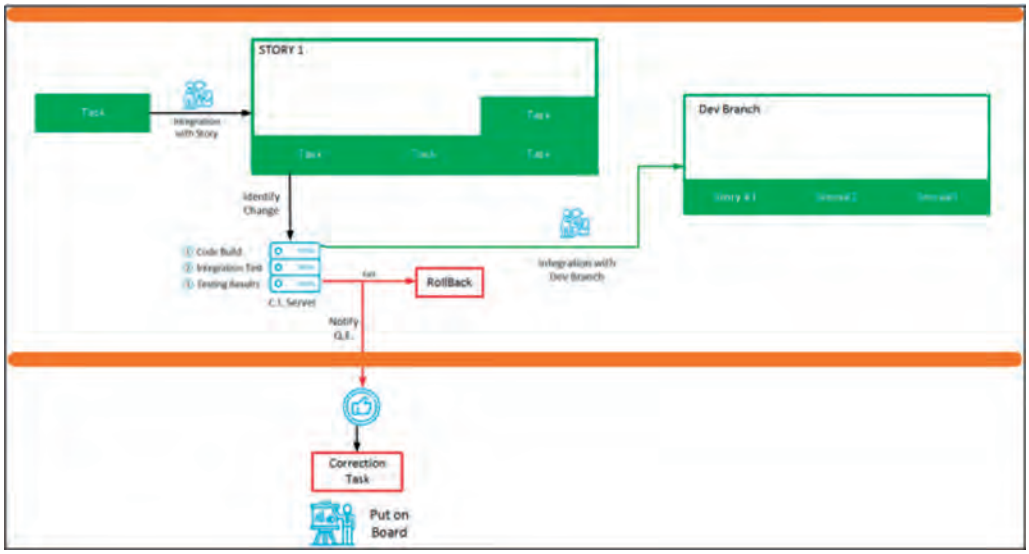


Figura 27. Integración continua del pipeline de desarrollo

de pruebas, y si las pruebas resultan correctas, el ingeniero de calidad realiza unas últimas pruebas junto con el equipo de proyectos, para validar la nueva versión de la aplicación o servicio. Una vez aprobada la nueva versión por los equipos de calidad y proyectos, se realiza la mezcla con la rama máster (producción) y se compila y presenta la nueva versión. Esta última fase del pipeline se presenta en la Figura 28.

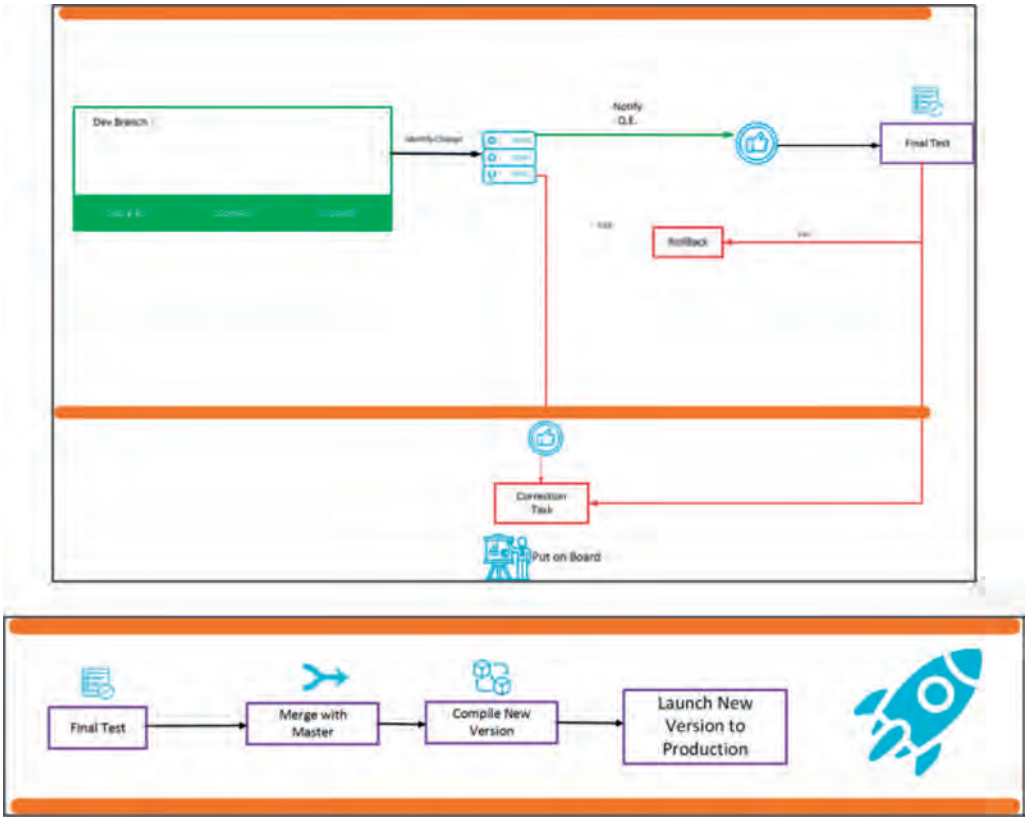


Figura 28. Última fase del pipeline de desarrollo

IMPLEMENTACIÓN DEL MODELO OPERATIVO ÁGIL

Roles, requerimientos y tareas

Previo a la implementación y apropiación del modelo operativo se realizó un taller con un enfoque de aprendizaje basado en problemas [PBL, *Problem*

Based Learning], para él se preparó un caso de estudio ajeno a los proyectos manejados en la empresa que permite aplicar el modelo operativo de manera controlada, a través de un ejercicio académico que con su didáctica facilita la apropiación del conocimiento. El caso considera el desarrollo de una aplicación web adaptable para el manejo de viáticos en una universidad, en él se plantean todos los usuarios que interactúan con el sistema y algunos requerimientos intencionalmente poco detallados –para propiciar un ambiente de dudas respecto del producto final y que los asistentes tengan que realizar entrevistas y plantear preguntas al cliente (el director del taller)–, útil para destacar tanto la importancia del involucramiento del cliente como de los principios ágiles. Además del caso, se realizó una presentación para afianzar el conocimiento y generar conclusiones respecto del trabajo realizado. El caso de estudio y el taller de apropiación son un ejercicio pedagógico para que los colaboradores aprendan el manejo de historias de usuario y tareas.

Es importante precisar que la formalización de las áreas organizacionales que propone el proyecto es responsabilidad de la gerencia. Para efectos del taller, se dividió a los colaboradores en grupos de tres personas y se les asignó roles específicos: proyecto, desarrollado e infraestructura. Así, pues, todo el desarrollo del taller se realizó con los roles y responsabilidades claras para cada persona; los grupos se formaron con personas de diferentes equipos y se asignaron roles diferentes al que tendría en su día a día, con el fin de generar empatía entre los colaboradores, que todos entiendan las responsabilidades de cada rol.

Como se ha expuesto, los requerimientos son el elemento fundamental para organizar el trabajo en la empresa. Durante los talleres, el proceso para su manejo como historias de usuario inició con una reflexión sobre el modelo operativo actual; luego se planteó la construcción de historias de usuario para el caso de estudio y se generó una discusión respecto de ellas; por último, se realizó el desglose de las historias seleccionadas en tareas.

Para la discusión reflexiva del proceso actual de manejo de los requerimientos, cada grupo debía describir y exponer cómo se hace en la empresa, con la expectativa de llegar a un consenso. En la discusión se encontraron respuestas que muestran su malestar respecto del proceso actual:

> ... los requerimientos son asignados por completo a cada uno, y, a veces, son tan grandes e incluyen tantas cosas que uno no sabe por dónde empezar... además, son temas complejos que toman mucho tiempo

> ... a veces llegan como una frase o serie de frases como: "se necesita un nuevo módulo para portafolio", pero no se especifica bien
>
> ... en ocasiones no es posible desarrollar un requerimiento de manera seguida porque llegan otras cosas y otros requerimientos ... es un poco difícil volver a tomar el hilo.

El consenso construido con base en su percepción presenta diferencias con lo expuesto en la formalización de los procesos de desarrollo de requerimientos (validados con los gerentes). Durante la actividad, en una discusión reflexiva, los colaboradores identificaron oportunidades de mejora. La principal dificultad radica en que no conocen cómo aplicar esas oportunidades ni cómo lograr una mejora real del proceso.

Luego de obtener el consenso, se presentó el proceso a desarrollar y se mostró cómo las oportunidades de mejora identificadas se pueden solucionar con la aplicación del modelo operativo propuesto. Se presentó el caso de estudio y se inició con el trabajo pedagógico. Cada grupo leyó y analizó el caso y estimó los posibles requerimientos de la aplicación. Estas aproximaciones a los requerimientos fueron la base para iniciar la migración al manejo de requerimientos como historias de usuario.

A cada grupo se le solicitó escribir en una cartulina por lo menos una historia de usuario para cada uno de cinco tipos de usuario propios del caso, siguiendo la estructura del "yo como: ____", "quiero: _____", "para: ____". Aquí el ejercicio se tornó interesante, los integrantes mostraban dificultades para formular requerimientos desde la visión del usuario, lo que generó confusión respecto de qué es la historia de usuario y qué es un requerimiento. Además, las expresiones "quiero: ___" y "para: ___" fueron difíciles de expresar pues consideraban que significaban lo mismo. Esta discusión generó una disonancia cognitiva en los colaboradores debido a que rompieron sus ideas previas de qué debía ser un requerimiento y cómo se debe especificar, con lo que se logró el objetivo principal de aplicar una metodología de aprendizaje basada en PBL.

En un principio los desarrolladores asumieron que un requerimiento era la necesidad de alguna persona y que ellos debían especificar todo el desarrollo. Ahora, con las historias de usuario, los requerimientos eran más personales e identificaron la importancia de definir un objetivo para el requerimiento. Cada grupo presentó sus historias de usuario, todos concordaron en que no es suficiente con solo tener la historia.

Se solicitó escribir en la parte posterior de las tarjetas las restricciones y validaciones de la historia de usuario y así presentar el detalle completo de la historia y despejar todas las dudas sobre el requerimiento del cliente. En este punto, el facilitador del taller se convirtió en el principal interesado de los equipos, el cliente, con lo que ellos experimentaron la importancia de su involucramiento y retroalimentación constante. Resultó interesante observar el malestar de algunos al realizar el ejercicio, ya que los datos adicionales provistos por "el cliente" contrastaban con sus supuestos, lo que les obligaba a modificar lo escrito en las historias de usuario. Esta situación fortaleció la percepción de la importancia de involucrar al cliente en el proceso. Cuando los grupos terminaron de corregir sus historias, se les solicitó escribir las historias definitivas en otra tarjeta.

El siguiente ejercicio consistió en compartir las historias de usuario con los otros grupos, de tal manera que cada persona tuviera una diferente de las que se trabajaron en su grupo. Cada uno debía leer la historia en voz alta y explicar cómo la entendía; algunas historias quedaron claras, otras dieron pie a discusiones respecto de la interpretación de cada persona. Este punto fue perfecto para explicar las cuatro reglas básicas de todo requerimiento [54]: deben ser simples, completos y bien estructurados, sin conectores (e.g., y, sin embargo, pero); no se debe especificar cómo se va a dar solución al requerimiento, deben quedar por fuera las tecnologías, arquitecturas y demás consideraciones de TI; deben estar dentro del alcance y apuntar a solo un componente, característica o funcionalidad; y se debe eliminar la ambigüedad.

De esta manera, el grupo concluye en la importancia de ser claro en la construcción de las historias y contemplar todos los detalles necesarios, no obviar elementos –como el tipo de datos de un campo en un formulario–, y poner todos estos detalles en las validaciones; aprecia también cómo al tener un mayor detalle en las historias, las validaciones van a ser más completas y menor el número de NC, porque se va a desarrollar lo que desea el cliente.

Si bien el grupo concluyó que las historias de usuario son una manera más empática y precisa para el desarrollo de requerimientos, también piensan que asignar una historia de usuario por ingeniero es un procedimiento extenso y difícil para una sola persona. Aquí, las tareas cobran un sentido especial, debido a que son actividades atómicas que presentan una menor dificultad y disminuyen el tiempo de desarrollo. Esto, junto al uso de repositorios de código, permite la construcción colaborativa de una historia de usuario y así,

no es una persona la que construye la historia sino el equipo de desarrollo, a través de la realización de tareas paralelas. Por esto, la claridad de las tareas y su manejo es clave.

Utilizando las historias de usuario construidas, se desarrolló un ejercicio de definición de tareas para cada historia. El facilitador expuso la definición de una tarea y explicó la relación entre ellas mostrando el uso de tareas secuenciales y paralelas. La siguiente instrucción fue elaborar una propuesta con todas las tareas para desarrollar la historia de usuario. Aquí, se generó una situación interesante, la alta dependencia del equipo de desarrollo del equipo de infraestructura, ya que todas las tareas de desarrollo dependen de la correcta creación de la infraestructura con cada los recursos de AWS. Por ejemplo, una tarea repetida en todos los casos fue la creación de la tabla en base de datos, pero solo las personas del equipo de infraestructura contemplan la tarea completa de creación, aprovisionamiento y configuración de la base de datos.

Esta actividad permitió también dimensionar todas las tareas necesarias para desarrollar una historia de usuario y visualizar que ciertas tareas, como validar que un campo solo permita números y establecer conexión con la base de datos, pueden desarrollarse paralelamente, lo que reduce el tiempo necesario para completar las historias de usuario. Otro resultado interesante fue que las tareas de pruebas de usuario solo fueron consideradas en detalle por una persona del equipo de calidad y dos del equipo de proyectos, lo que muestra la falta de una cultura organizacional orientada a dar calidad a las soluciones desarrolladas, en la que se da más énfasis a las pruebas unitarias que emplea cada desarrollador.

Como conclusión de este ejercicio, los equipos determinaron: que un trabajo colaborativo por tareas permite una entrega rápida de historias de usuario que representan nuevas características o funcionalidades en una aplicación o servicio; que es muy importante realizar pruebas a todas las validaciones definidas en la historia de usuario para disminuir las NC; y que en un modelo ágil, no es necesario esperar hasta que la aplicación o servicio se encuentre desarrollada en su totalidad para la entrega de valor al cliente, que es posible realizarle entregas tempranas de la aplicación o servicios y así darle valor constantemente, siempre mencionándole que se está realizando este proceso para que no se generen NC que afecten los resultados de la compañía en general. Lo que se busca es un proceso de retroalimentación constante que le muestra al cliente, en todo momento, los avances, para conocer su opinión e

identificar oportunidades de mejora en etapas tempranas del desarrollo de la aplicación o servicio.

Concluido el taller, llegó el momento de la implementación de las historias en el trabajo diario de la compañía. Para facilitar esta actividad se seleccionó Trello, una herramienta web y móvil de colaboración que organiza proyectos en tableros [55] y opera con la base de un tablero en blanco que se llena de listas de notas adhesivas. En él, cada nota es una tarea, requerimiento, actividad u objetivo para el equipo, cada nota tiene fotos, archivos adjuntos de otras fuentes de datos –como BitBucket o Salesforce–, documentos y un lugar para comentar y colaborar con los miembros del equipo [56]. Cabe mencionar que en la empresa ya se usa Trello para el manejo de los casos de soporte, por lo que los usuarios están familiarizados con la herramienta y su manejo.

Se creó un tablero denominado con el nombre de la empresa en el cual estarán todas las historias de usuario de la compañía. Para organizar dichas historias, se utilizó el modelo Kanban, el cual fomenta un enfoque de gestión de requerimientos, riesgos comerciales y capacidad de cada servicio [57] y plantea múltiples listas de trabajo. En este modelo operativo se manejan tres: la lista de lo que está por hacer (*To do*), la lista de lo que se está haciendo (*Doing*) y la lista de lo terminado (*Done*). Al equipo se le solicitó organizar el tablero y las historias de usuario.

Antes de empezar a agregar las historias de usuario se creó un equipo de tablero en Trello compartido por todos, de esa manera todos pueden modificarlo y pueden revisarlo juntos. Cada persona empezó a completar las historias de usuario considerando lo aprendido durante el taller. En el título de la tarjeta, se escribe el ID de la historia y un título para la misma; en la descripción se escribe el objetivo de la historia siguiendo el esquema del "Yo como...", "quiero...", "para...". Además, se crea una lista de chequeo dentro de la tarjeta para el manejo de las restricciones y validaciones de la historia. Este es un plugin importante que ofrece Trello, el cual permite estimar un avance de la historia de usuario en la medida en que se van cumpliendo sus restricciones. Por último, se especifican los miembros que trabajan y son responsables de cumplir con la historia de usuario, se etiqueta la historia según el proyecto al que pertenece y se establece una fecha de entrega estimada.

Con la construcción completa de las historias de usuario, el equipo de proyectos y la gerencia pueden ver en un solo tablero: los requerimientos que están pendientes; y las historias que se están trabajando, su estado de avance y

las fechas de entrega; además, pueden medir a los equipos al revisar las historias que se están terminando.

Un último elemento importante fue la integración entre el tablero y la aplicación de comunicación colaborativa Slack [58], un centro de colaboración que puede reemplazar el correo electrónico y permite a los equipos trabajar juntos sin problemas. Slack se divide en espacios de trabajo conformados por canales, donde los miembros del equipo pueden comunicarse y trabajar juntos, enviar mensajes y compartir archivos en los canales. Es posible crear canales para equipos, proyectos, sucursales de la empresa o para cualquier otro aspecto que sea pertinente para la organización [59]. Con la integración de Trello y Slack:

- se comparte la información con el equipo y se obtienen comentarios sobre un proyecto enviando tarjetas Trello a los canales de Slack o mensajes directos;
- se configuran las alertas para notificar automáticamente a los canales de Slack sobre la actividad en Trello a nivel de tablero, lista y tarjeta;
- se obtienen los recordatorios para las tarjetas, para cuando las necesite en Slack o para establecer recordatorios para otro compañero de equipo, para que nunca se olviden las historias importantes; y
- se convierten las tormentas de ideas de Slack en elementos accionables en Trello, uniendo la conversación de Slack a las tarjetas.

La Figura 29 muestra un ejemplo de la tarjeta Trello como historia de usuario para la empresa.

Si bien luego de una semana de trabajo las historias de usuario empezaron a convertirse en un elemento fundamental de las labores de la empresa y se empezaron a apreciar los buenos resultados de la propuesta del modelo operativo, es claro que no es tiempo suficiente para su apropiación; por ello, se hizo un trabajo de refuerzo durante dos semanas más en la utilización de las historias de usuario y en la migración hacia la nueva forma de manejar los requerimientos. En la Figura 30 se aprecia un ejemplo del manejo final de las historias de usuario en las labores de la compañía.

Tablero de priorización

Como se estableció en el modelo, el siguiente paso es determinar las tareas corporativas que serán establecidas en el tablero de priorización. Cuando se inició su implementación, generó rechazo desde los equipos de proyectos y

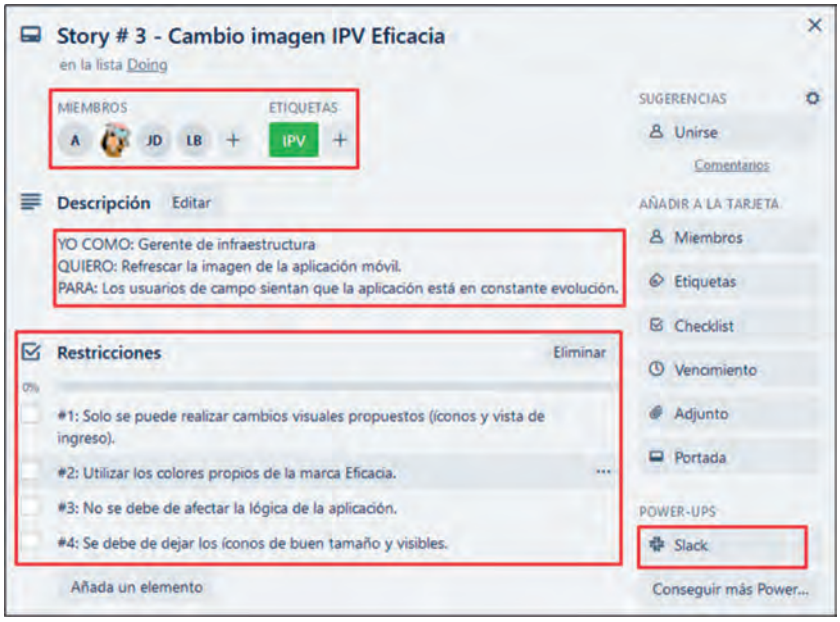


Figura 29. Tarjeta Trello como historia de usuario

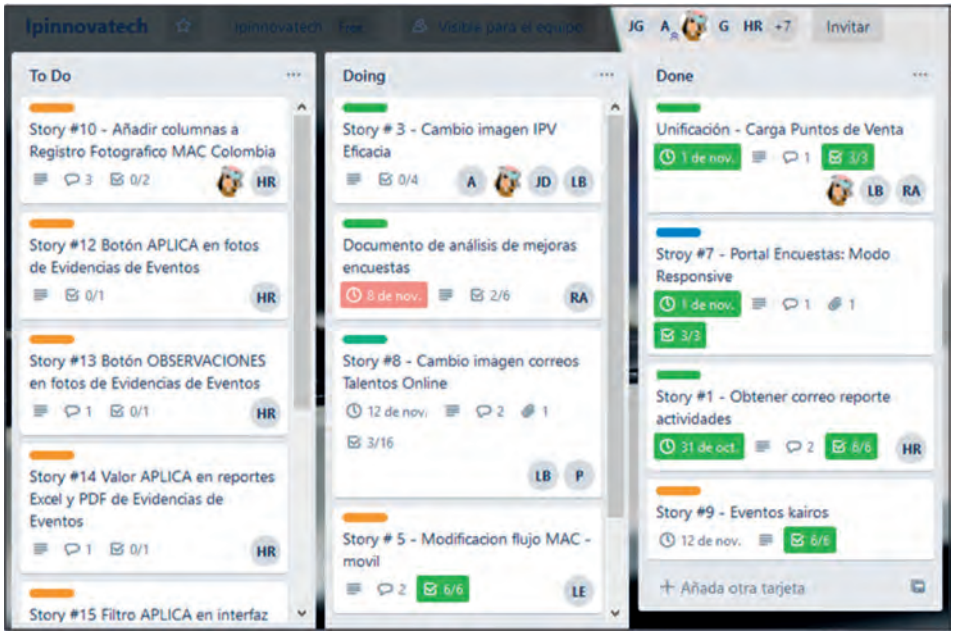


Figura 30. Manejo final de las historias de usuario

desarrollo, ya que tradicionalmente era la líder de proyectos quien asignaba el trabajo a cada persona; con el tablero, las tareas quedan organizadas y es trabajo de cada miembro de los equipos tomar una tarea prioritaria e iniciar su desarrollo.

Al equipo de proyectos, al iniciar la implementación del tablero, le resultó difícil establecer las tareas atómicas, ya que las tareas en un principio eran grandes, complejas y contemplaban muchas actividades para ser completadas, y reducirlas a elementos y procesos más pequeños es un trabajo que día a día está formulando el equipo de proyectos. Al cierre de este documento, el equipo de proyectos continúa trabajando en dejar las tareas más atómicas para el equipo de desarrollo; por el momento, el equipo asume entender las tareas como las validaciones o restricciones de cada historia de usuario, lo que es una práctica aceptable para el equipo y ayuda a una apropiación más amigable del modelo.

El equipo de proyectos tuvo una segunda gran dificultad, priorizar las tareas, porque todas al principio parecen igual de prioritarias y todas tienen una entrega en la misma semana. Esto provocó que las tareas en su totalidad fueran ubicadas en el primer cuadrante del tablero, lo que generó confusión y estrés, pues todas las tareas eran prioritarias. En este punto, se acompañó al equipo de proyectos ayudándole a analizar cada una de las tareas, para ubicarlas lo mejor posible en el tablero.

El primer paso fue organizar en una línea de tiempo la fecha de entrega de cada tarea, aquí, se evidenció que las tareas fueron programadas pensando en la entrega de la historia, es decir, todas las tareas de una misma historia presentaban la misma fecha de entrega, lo que no permitía una programación continua en toda la línea de tiempo. Restablecer las tareas, pensando en la semana completa, organizándolas por día de la semana, permitió ubicarlas en la línea de tiempo (horizontal); para organizarlas verticalmente se pidió establecer su criticidad según la escala presentada en la Figura 22, para luego ubicarlas en el tablero según la nueva relación, de esa manera, las tareas quedaron repartidas en el tablero y se eliminó que todas muestren la misma prioridad.

Otra dificultad que se presentó con el equipo de proyectos fue el seguimiento del trabajo, ya que previamente los requerimientos eran asignados en su totalidad a cada ingeniero y el avance del proyecto se realizaba según los avances de cada uno. Este nuevo esquema dificulta realizar el seguimiento

a los ingenieros porque cada uno desarrolla diferentes tareas para diferentes proyectos y diferentes historias. En este punto, se refuerza el hecho de ver el equipo de desarrollo como un todo y que el equipo de proyectos debe velar por el avance constante de las historias de usuario. Así pues, el seguimiento deja de ser un trabajo personal y se vuelve un seguimiento de los proyectos e historias de usuario, lo que se logra revisando las confirmaciones [*commits*] por cada repositorio de la historia. Además, si se necesita priorizar una historia sobre otra, solo se debe modificar el orden de las tareas en el tablero. Como una experiencia de aprendizaje práctica, se recomendó al equipo de proyectos que su primera actividad de la semana sea actualizar el tablero, revisar las prioridades, los tiempos de entrega y la ubicación de las tareas, para tener así la planeación de la semana organizada. De esa manera, los equipos de desarrollo e infraestructura sabrán cómo priorizar.

Al iniciar el proceso de apropiación del tablero, los ingenieros del equipo de desarrollo estaban seleccionando principalmente tareas secundarias, debido a la facilidad de su realización; es clara entonces la necesidad de que el equipo de proyectos fomente la selección de las tareas prioritarias, para avanzar en los proyectos, considerando los tiempos y necesidad del cliente.

Para los equipos de desarrollo e infraestructura fue difícil comprender que las tareas no iban a ser asignadas por un jefe o líder de proyecto, sino que era necesario realizar un trabajo responsable al seleccionar las tareas del tablero. Los integrantes de ambos equipos manifestaron sentir menos presión, porque no están desarrollando solos un objetivo muy grande, sino que van desarrollando tarea por tarea, lo que les ayuda a cumplir más seguido su trabajo, y a estar involucrados en el desarrollo de la mayoría de las historias de usuario y comprender con mayor detalle cada proyecto. La dificultad más grande se presentó al identificar las tareas secuenciales y su manejo, ya que en un principio se asumió que la persona que inicia la secuencia debía tomar siempre dichas tareas. Por esto, se realizó un trabajo de reflexión con los equipos mostrando que el desarrollo de ciertas tareas, así sean secuenciales, se pueden pensar como un parámetro de entrada, organizando cada tarea como un método independiente. En este punto ocurrió algo interesante, el mismo equipo determinó los límites para esta práctica, al concordar en que no es viable desarrollar métodos con una gran cantidad de parámetros.

Si bien la utilización del tablero aumentó la colaboración entre los equipos, aún queda mucho camino por recorrer, lo dicho aquí es solo un primer

acercamiento a la migración hacia un modelo operativo ágil, por lo que es tarea de la organización fomentar el uso del modelo y continuar su aplicación y mejoramiento.

Implementación del pipeline de infraestructura

Este *pipeline* es la base para el equipo de desarrollo, ya que el equipo de infraestructura implementa, despliega y aprovisiona en él todos los elementos necesarios para que se pueda iniciar el desarrollo de las aplicaciones y servicios. El proceso de creación y configuración de una nueva infraestructura es un proceso manual que toma mucho tiempo, para automatizar dicho despliegue se utiliza IaC, que se puede desarrollar, como se explicó, con una de las dos herramientas líderes del mercado: Terraform y CloudFormation.

Se decidió usar CloudFormation por su alta compatibilidad con AWS y porque su herramienta CloudFormation Designer permite construir la plantilla de despliegue y aprovisionamiento con un entorno gráfico amigable (en la Figura 31 se puede apreciar cómo se establece el diseño de una infraestructura en AWS CloudFormation Designer). Para este caso, se construyeron cinco plantillas base para el despliegue de infraestructura en AWS, las cuales se presentan en la Tabla 6.

Esta última es la plantilla principal para cualquier despliegue, por su relevancia, se incluye una descripción de su ejecución. Primero, se debe descargar del repositorio y cargarla en el servicio AWS CloudFormation (ver Figura 32);

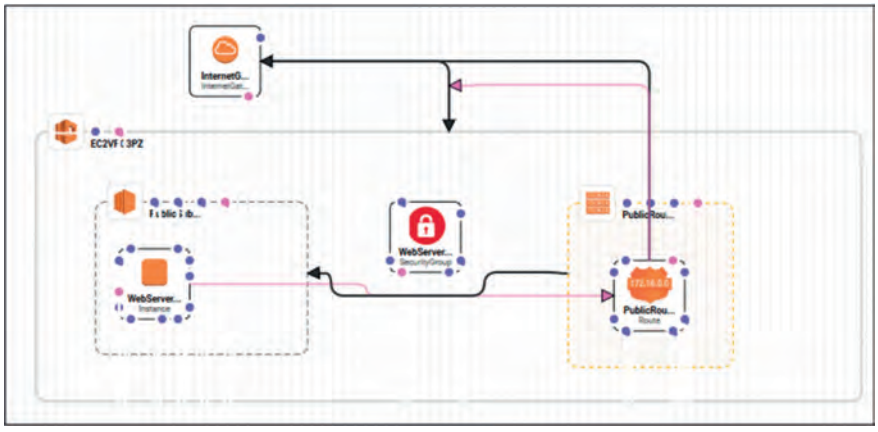


Figura 31. Arquitectura AWS CloudFormation

hecho esto, se deben configurar los parámetros para elegir un nombre para la pila de recursos, la VPC donde será implementado el servidor, el tamaño de la instancia, la llave de acceso al servidor, la etiqueta de proyecto, las conexiones SSH y el grupo de seguridad donde será implementada la instancia y la subred;

Tabla 6. Planillas construidas con CloudFormation Designer

Plantilla	Descripción
Automatization-Role-CloudFormation	Plantilla para crear los roles de IAM para EC2 y SSM útiles para permitirle al SSM gestionar de manera automática las instancias EC2.
OnlyVPC-CloudFormation	Plantilla para crear solo una VPC, una ruta pública y una puerta de enlace a Internet.
WebServer-in-VPC-CloudFormation	Plantilla básica para la creación de un web server con acceso público: VPC, Internet gateway, subred y grupo de seguridad y los parámetros tipo de instancia, keypair e IP para configurar Security Group.
WebServerinVPC-Demo	Plantilla de demostración que implementa un web server con una aplicación web de ejemplo en menos de cinco minutos.
WebServer-SelectVPC-DevTestclone	Plantilla que despliega infraestructura seleccionando todo y crea un stack LAMP [Linux Apache MySQL PHP] usando una instancia de EC2. Los usuarios eligen los parámetros de VPC, *security group*, *keypair*, nombre y etiquetas. Además, se crea un *bucket* S3y una copia del servidor desplegado en una VPC de desarrollo y pruebas.

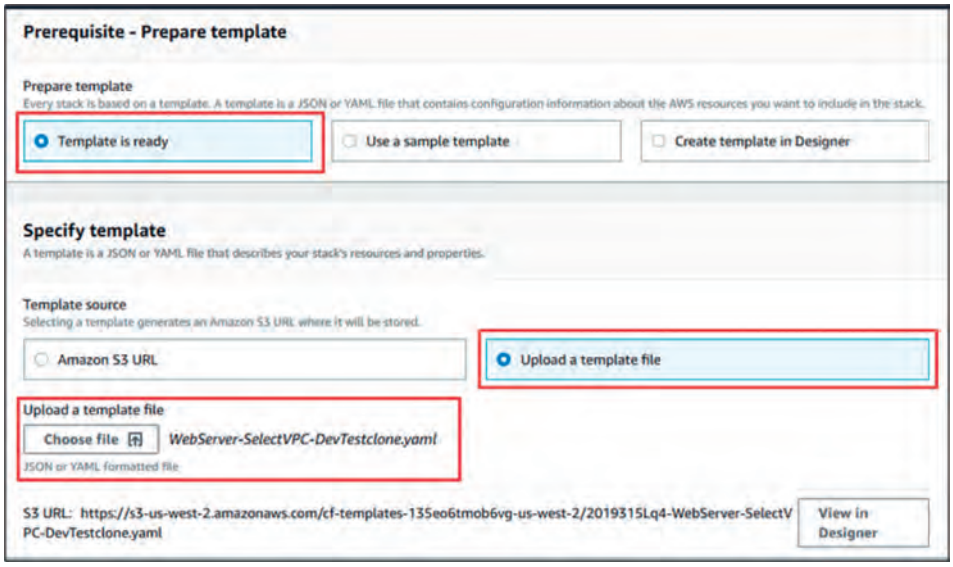


Figura 32. Carga de plantilla en CloudFormation

con esta información, la plantilla crea un servidor de pruebas para el nuevo proyecto en la VPC Dev/Test. AWS crea además un nuevo *bucket* S3 para guardar la plantilla, con lo que se puede utilizar en un futuro de manera muy rápida. Este enlace se guarda para futuras descargas y modificaciones de la plantilla. El proceso se ilustra en la Figura 33.

Figura 33. Parámetros de plantilla en CloudFormation

Una vez configurados los parámetros, se procede con la creación de la pila de recursos con las configuraciones por defecto de CloudFormation. Al iniciar ese proceso, AWS con CloudFormation ejecuta todos los comandos que se encuentran en la plantilla para implementar y configurar los recursos de AWS; adicionalmente, una vez que están en ejecución las instancias EC2, AWS CloudFormation aprovisiona de manera automática cada una de las instancias según un script cargado en la plantilla, en la sección del UserData. Este procedimiento puede tomar entre cuatro y ocho minutos, lo que implica una mejora significativa en el tiempo requerido para implementar, desplegar y aprovisionar los recursos de infraestructura necesarios para el desarrollo y puesta en operación de las aplicaciones o servicios que ofrece la compañía.

En la Figura 34 se aprecia el momento durante el cual AWS está implementando los recursos de la infraestructura de manera automática. AWS notifica cada proceso que se está realizando y la correcta ejecución de la plantilla; si existen errores en la ejecución, AWS detiene la creación de los recursos y realiza una tarea de retroceso [*rollback*], eliminando toda la infraestructura que se había implementado hasta el momento (Figura 35).

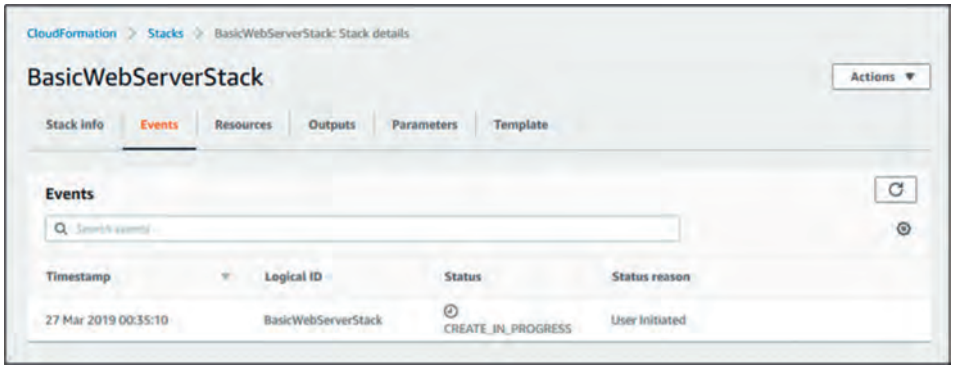


Figura 34. Implementación infraestructura en AWS CloudFormation

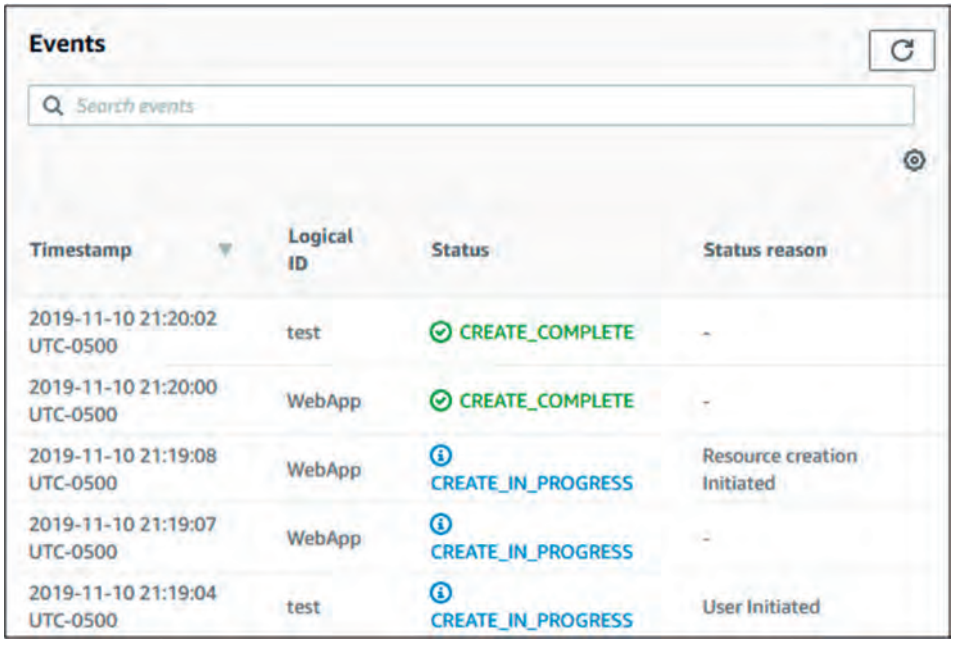


Figura 35. Eventos en CloudFormation

Como parte de las actividades del proyecto, se construyó una guía de introducción a CloudFormation e IaC [60] para que el equipo de la empresa pueda desarrollar sus propias plantillas o resolver dudas.

Implementación del pipeline de desarrollo

Como se explicó, este pipeline se encarga de implementar las prácticas continuas, aunque puede operar sin necesidad de la infraestructura, solamente con ella implementada, configurada y desplegada en AWS funciona sin contratiempos.

La implementación de este pipeline inicia con la creación de los repositorios de código con la herramienta Git, la cual permite el manejo de versiones de una aplicación. Luego de evaluar GitHub y CodeCommit se decidió utilizar el segundo por tratarse de repositorios seguros, privados, cuyo acceso se puede gestionar a través del servicio IAM. Además, CodeCommit permite la construcción de flujos de prácticas continuas con servicios integrados dentro de la misma herramienta.

CodeCommit cuenta con cinco servicios configurables que proveen una mejora en los repositorios de código: origen, compilación, despliegue, canalización y configuración. El servicio de origen es la base, es el encargado de crear los repositorios de código; el servicio de compilación –CodeBuild–se utiliza para una integración continua completamente administrada por AWS, quien compila código fuente, ejecuta pruebas y produce paquetes de software listos para su implementación en ambientes productivos [61]; el servicio de despliegue –CodeDeploy–es un servicio completamente administrado por AWS que automatiza el despliegue de software en servicios de computo de AWS, como Amazon EC2, AWS Lambda y servidores locales, lo que facilita el lanzamiento rápido de nuevas características [62]; el servicio de canalización –CodePipeline– es un servicio de integración y entrega continuas para realizar actualizaciones de aplicaciones e infraestructura rápidas y confiables, compila, prueba y despliega el código cada vez que se produce un cambio, de acuerdo con los modelos de procesamiento de lanzamiento que se definan [63]; el servicio de configuración es utilizado para crear reglas de notificación que ejecutan los servicios descritos para informar a los ingenieros de los equipos de desarrollo y calidad sobre cualquier novedad.

La implementación inició con la creación de los repositorios en CodeCommit. Previamente se definieron las nuevas políticas del grupo de usuarios de

desarrolladores dentro de IAM, las cuales se centran en el acceso completo al servicio de CodeCommit. Luego se crearon los repositorios de código para cada uno de los proyectos y las ramificaciones para el desarrollo de cada historia de usuario. Cabe resaltar que el equipo de ingenieros identificó que la propuesta realizada en el modelo no era aplicable en la compañía debido a que muchas veces las nuevas características que se desarrollan nunca terminan en el ambiente de producción, por lo que es arriesgado juntar el desarrollo de todas las historias de usuario, características y funcionalidades en una sola rama de desarrollo que luego se mezcla con producción.

La organización de los repositorios y la apropiación del conocimiento en el manejo de Git fue un ejercicio totalmente proactivo de los equipos de proyectos y desarrollo, quienes crearon los repositorios y las ramas y aprendieron el manejo de los comandos en Git, para tener su código centralizado. Esto fue posible porque el equipo ya había identificado el valor del modelo operativo ágil y la importancia del trabajo colaborativo.

Como se explicó, el *pipeline* de desarrollo contempla la implementación de las práctica de CI y CDE en el modelo operativo. Para la implementación de las prácticas continuas, se utilizan los repositorios creados en CodeCommit, un servidor de integración con el sistema Jenkins, el servicio S3 de AWS y el servicio de despliegue CodeDeploy en AWS. Jenkins es un servidor de automatización de código abierto autónomo que se puede utilizar para automatizar todo tipo de tareas relacionadas con la creación, prueba y entrega o implementación de software [64]. En la Figura 36 se evidencia el proceso de implementación general de la práctica continua. El desarrollo se realizó con base en [65], guía que plantea la construcción de una tubería, dividida en tres trabajos: identificar un nuevo *push* en los repositorios de CodeCommit y construir un artefacto con el código fuente; ejecutar las pruebas de integridad que validan el artefacto por completo, verificando que todos los componentes del software se encuentren bien integrados y se pueda desplegar una nueva versión de la aplicación o servicio (si las pruebas de integridad son exitosas, Jenkins guardará el artefacto como una nueva versión de la aplicación o servicio en un *bucket* de S3); y ejecutar una tarea de CodeDeploy que despliega el artefacto validado en una instancia de EC2 con un proceso automatizado. De esta manera, se logra entregar al cliente una nueva versión del aplicativo. En el caso de fallos en cualquier trabajo, se envía un correo notificando al equipo de calidad. Antes de iniciar la creación los trabajos, es necesario crear en AWS los recursos incluidos en la Tabla 7.

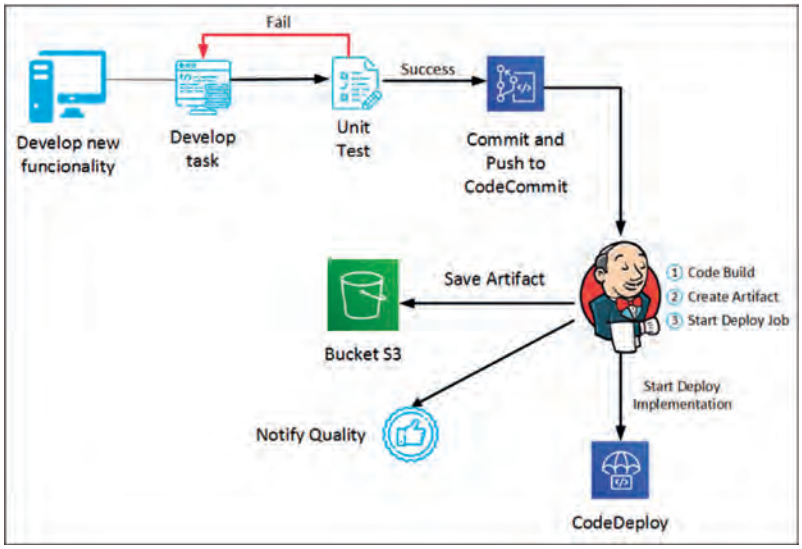


Figura 36. Proceso de integración continua

Tabla 7. Recursos requeridos en AWS

Recurso	Descripción
Repositorio CodeCommit	Para que todo el pipeline tenga sentido, el código debe estar en un repositorio en la herramienta Git.
Buckets S3	Se deben crear dos *buckets* con el servicio de S3 para almacenar los artefactos, los cuales deben tener habilitado el versionamiento de archivos. Para la construcción del primer pipeline, que sirve de ejemplo para los equipos de la empresa, se crearon dos *buckets*: ci-bucket-code y ci-bucket-jenkins: el primero para almacenar el código de la aplicación compilado después del primer trabajo; y el segundo para almacenar el artefacto valido de la aplicación, listo para su despliegue en la instancia EC2.
Roles	Su configuración es necesaria porque existe comunicación entre diferentes recursos de AWS –instancias, *buckets* y code deploy–. Se crearon dos roles: el primero permite la comunicación desde una instancia EC2 a un *bucket* S3, así, Jenkins puede enviar los códigos compilados y los artefactos a los *buckets*; el segundo le permite al servicio de CodeDeploy consultar los artefactos de los *buckets* de S3 y desplegar el artefacto dentro de la instancia EC2 de la aplicación o servicio.
Grupo de seguridad	Se debe crear un grupo de seguridad que permita la conexión a la instancia EC2 de Jenkins a través de los puertos 22, para SSH, y 8080, para el ingreso a la herramienta web de Jenkins.
Instancia EC2	Se debe crear para la ejecución de Jenkins, en este caso se creó una instancia con sistema operativo RHEL con tamaño t2.micro.

Tabla 7. Recursos requeridos en AWS (cont.)

Recurso	Descripción
Aplicación en CodeDeploy	Dentro de CodeDeploy, se debe configurar una aplicación que contiene uno o varios grupos de implementación. Cada grupo determina sobre qué instancias se va a desplegar el artefacto y cómo se realizará dicho despliegue. Los despliegues se pueden hacer a una instancia, a un grupo de auto escalado o a una instancia on-premises (ubicada en un servidor local del cliente). Los despliegues se pueden realizar: en el lugar –actualizar las instancias en el grupo de implementaciones con las últimas revisiones de la aplicación– o azul/verde –durante una implementación, cada instancia se desconecta brevemente para su actualización, mientras que azul/verde sustituye a las instancias en el grupo de implementaciones con instancias nuevas y les implementa la última revisión de la aplicación–. Después de que se registran las instancias en el entorno de sustitución con un balanceador de carga, se anula el registro de las instancias del entorno original y se pueden terminar. Este grupo de implementación ejecuta Jenkins.

Una vez creados todos los recursos necesarios para la aplicación de prácticas continuas, se debe: configurar la instancia EC2, aprovisionar la instancia con los paquetes de software necesarios por Jenkins (java y node.js) y configurar Jenkins para su acceso vía web. Esta actividad se realiza con un *script* en *shell*. Una vez configurado Jenkins, se accede vía web a través de la IP de la instancia EC2 y el puerto 8080 y se configuran las credenciales de acceso. Jenkins permite la instalación de *plugins* que facilitan la integración y automatización de los trabajos que van a ser ejecutados. Adicional a los *plugins* que recomienda Jenkins instalar en la primera ejecución del servicio, se deben instalar: S3 Publisher Plugin, Build Pipeline y AWS CodeDeployPlugin for Jenkins.

Una vez se ha implementado y aprovisionado todo el ambiente necesario, se puede iniciar la construcción del *pipeline* de automatización. Antes de empezar con la construcción de cada trabajo fue necesario configurar las llaves para acceso a la cuenta de AWS para S3 y configurar Jenkins para que AWS confié en las claves HTTPS del repositorio Git. Con estas configuraciones Jenkins puede ejecutar tareas sobre el repositorio y sobre los *buckets* de S3.

De los trabajos citados, el número 1 se centra en realizar una copia del código en CodeCommit y compilar el código para hacer un *build*. Se configura la conexión con el repositorio en CodeCommit y, al ser la tarea inicial del *pipeline*, se configura el lanzador ante cualquier cambio en el repositorio (ver proceso en la Figura 37); luego, se configuran los comandos que Jenkins va a

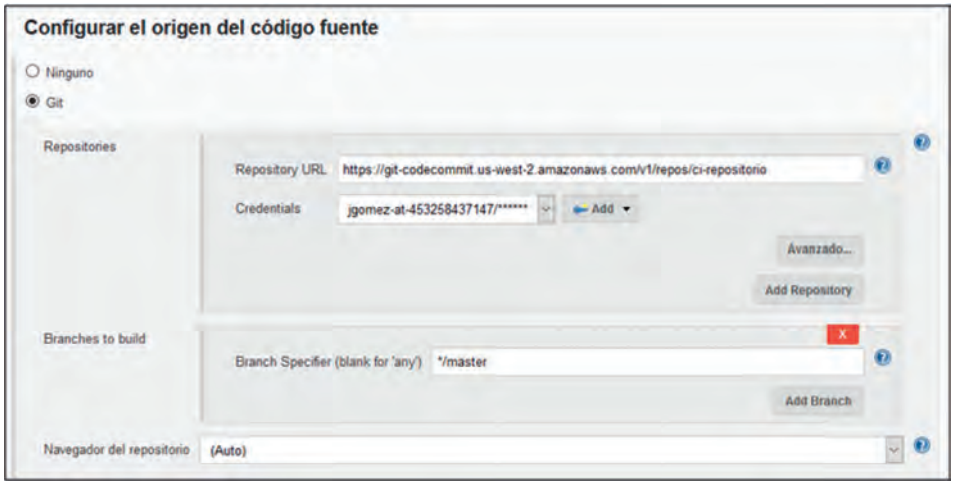


Figura 37. Configuración conexión repositorio CodeCommit

ejecutar cuando inicie la tarea, aquí se maneja la ejecución de dos comandos para realizar el *build* de la aplicación o servicio: npm install y npm run build. Posteriormente, se ejecutan las tareas que van a ocurrir después de construir el artefacto; se especifica que se deben guardar todos los archivos en el bucket de S3 y se especifica la siguiente tarea en el *pipeline*; y por último, se configura una notificación por correo electrónico (Figura 38).

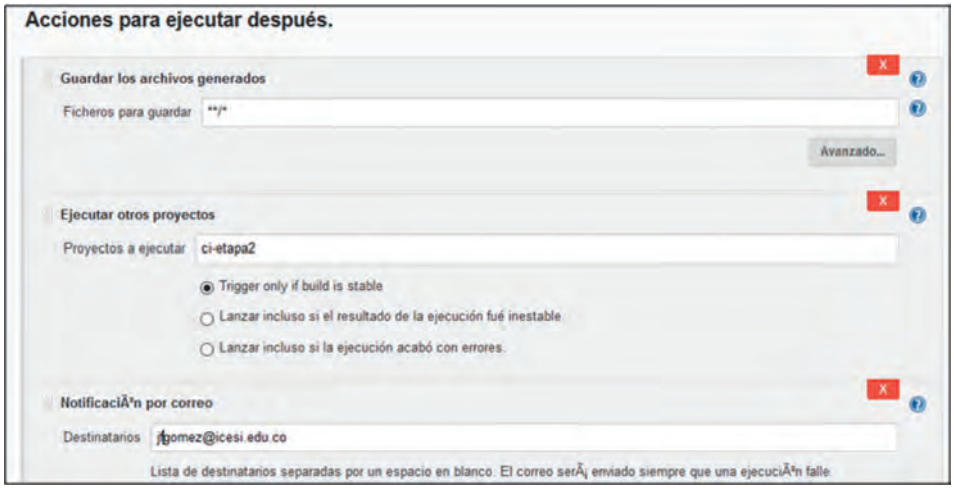


Figura 38. Acciones posteriores a la tarea 1

El Trabajo 2 toma el artefacto construido por el Trabajo 1 y lo guarda en el bucket S3. En él: se especifica el perfil de conexión, el nombre del bucket y la región de AWS; se configura el lanzador para el Trabajo 3; y se envía la notificación por correo. El artefacto solo se guarda si la construcción del Trabajo 1 tiene un resultado estable. En la Figura 39 se aprecia la configuración de la ejecución que guarda el artefacto en S3.

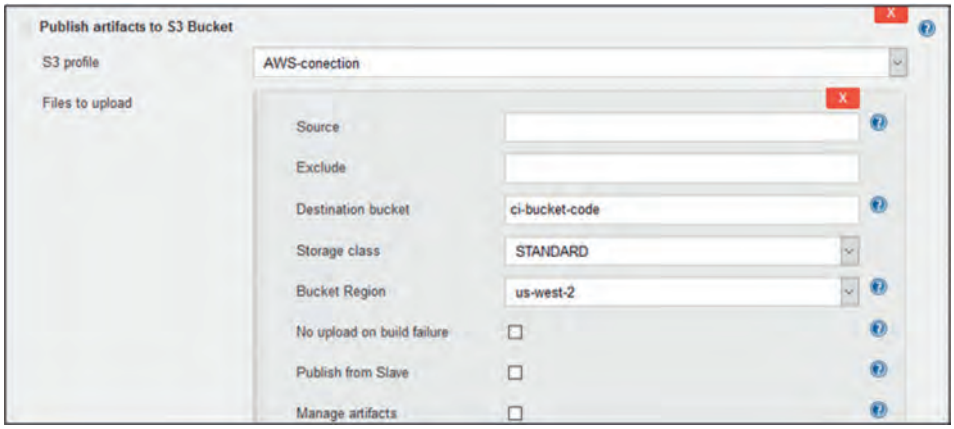


Figura 39. Tarea para guardar artefacto en S3

El Trabajo 3 toma el artefacto construido por el Trabajo 2 –que se encuentra en los bucket de S3–, genera el paquete completo de la aplicación en un archivo .zip listo para su despliegue en la instancia EC2 de la aplicación o servicio, y guarda el archivo .zip en el bucket S3. Luego Jenkins, utilizando la aplicación y el grupo de implementación configurado en CodeDeploy, le pasa el archivo .zip para que este se encargue del despliegue de la aplicación en la instancia EC2 (Figura 40).

Con los tres trabajos definidos es posible construir el pipeline con la configuración de un nuevo panel. En este panel se especifica cuál es la tarea que inicia el pipeline de automatización y de manera automática Jenkins muestra el pipeline por completo (Figura 41). El color verde en la figura indica que todo el flujo se ejecutó de manera correcta y que la aplicación está desplegada.

De esta manera finaliza la implementación del pipeline de desarrollo. A partir de ahí, el equipo de calidad puede definir las pruebas especializadas de

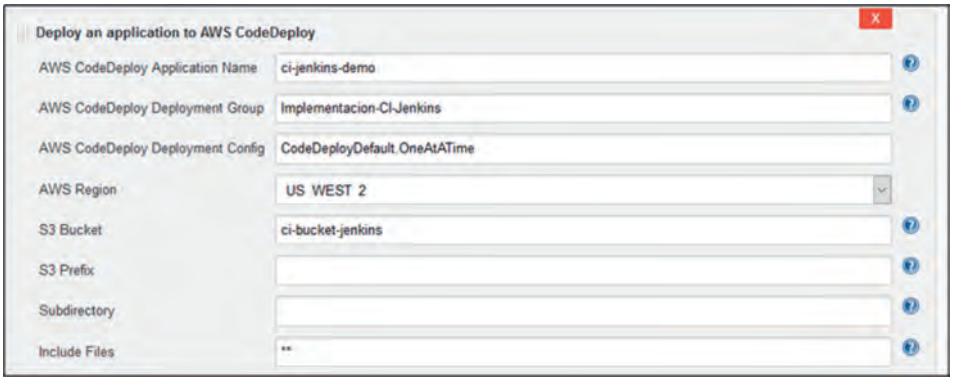


Figura 40. Configuración despliegue desde Jenkins a CloudDeploy

Figura 41. Pipeline en Jenkins

acuerdo con las características de cada uno de los servicios del portafolio de la empresa siguiendo un proceso de integración y entrega continuas, reduciendo el tiempo para brindar nuevas características y funcionalidades a sus clientes a través del pipeline de automatización.

CONCLUSIONES

- Realizar un proceso de análisis de negocio es de gran relevancia en la construcción y aplicación de un modelo operativo porque ayuda a identificar una oportunidad de mejora o la problemática principal de la compañía y sus causas, y así justificar la necesidad de cambio del modelo operativo.
- El modelo operativo permite agregar valor a las organizaciones, habilitando una mejora en la estrategia comercial y estableciendo una sinergia entre las personas, los procesos y la tecnología, lo que se evidencia con la materialización del concepto de Virtual Connect, como una sinergia entre todos los servicios de la compañía.

- Con el uso de las historias de usuario se logra que los equipos de la empresa, de manera natural, identifiquen que asignar un requerimiento completo por ingeniero es una práctica negativa y que es de suma importancia establecer tareas simples, fáciles de asignar y de trabajar.
- La planeación basada en tareas ayuda a aumentar el rendimiento del equipo y disminuye la frustración; su ejecución paralela facilita un trabajo colaborativo que propicia la construcción eficiente y ágil de aplicaciones y servicios de TI.
- El modelo operativo permite tener un alineamiento del negocio con tecnologías emergentes que permiten agregar valor a los servicios ofrecidos, en menos tiempo. Se evidencia una mejora en la puesta en funcionamiento de aplicaciones y servicios de la compañía al aplicar practicas continuas como CI y CDE.
- El modelo de diagnóstico como herramienta para identificar la percepción de las personas respecto de los procesos específicos de la empresa, su contraste y consolidación, es clave para priorizar la mejora de los procesos que generan mayor disonancia entre los colaboradores y gerentes. Con esta herramienta es posible también dimensionar la importancia de formalizar los procesos corporativos y compartirlos con los colaboradores para evitar dificultades en la operación diaria de la empresa.
- El modelo operativo propone una transformación de la cultura de trabajo en las PYME de la región, integrando los equipos de desarrollo y operación. El modelo propuesto en este proyecto considera varios mecanismos y metodologías ágiles, propone una migración amigable y óptima al agilismo y se ajusta a las necesidades de las PYME.
- Como resultado del trabajo realizado en la empresa, se evidencia una reducción en los tiempos de entrega de nuevas aplicaciones y servicios, y la oferta de nuevas características y funcionalidades para los servicios de TI existentes.
- En el aspecto tecnológico, se identifica que la falta de un diseño de la infraestructura adaptado a las necesidades de la compañía y la no aplicación de buenas prácticas de diseño provocan: sobrecostos, lentitud en el despliegue de nuevos servicios y mayor tiempo de salida al mercado. Un buen diseño de la arquitectura optimiza los costos, la seguridad y el rendimiento de la infraestructura, y logra aplicaciones y servicios estables que benefician al cliente y aumentan su satisfacción.

- La automatización de procesos con prácticas continuas, soportado con herramientas de última tecnología –como CloudFormation o SystemManager–, reducen el tiempo destinado a la implementación y gestión de la infraestructura y le permiten a los equipos de operaciones aprovechar el tiempo para innovar y mejorar las aplicaciones y servicios.
- Este es un proyecto centrado en la transformación digital y la innovación de la manera como operan las compañías de base tecnológica en el Valle del Cauca. Se espera que con la apropiación del modelo, diferentes organizaciones puedan obtener los beneficios de desplegar y aprovisionar sus servicios de manera ágil, y que ello aumente la competitividad de la región, con clientes satisfechos y soluciones innovadoras que producen valor.

REFERENCIAS

[1] S. Sharma, *The DevOps adoption playbook: A guide to adopting DevOps in a multi-speed IT enterprise*, Hoboken, NJ: Wiley, 2017.

[2] "Formación e innovación para el fortalecimiento de la competitividad del sector de las tecnologías de la información y comunicación (TIC) de la región: FormaTIC e InnovaTIC Valle del Cauca," Sistema General de Regalías, Bogotá, Colombia, 2019.

[3] PMI, *Agile practice guide*, Newtown Square, PA: PMI, 2017.

[4] International Institute of Business Analysis, *IIBA Global Business Analysis Core Standard*, Toronto, Canada, 2017.

[5] International Institute of Business Analysis, *BABOK Guide*, Toronto, Canada: IIBA, 2017.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, y J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, 2008.

[7] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, y S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14-76, 2014.

[8] P. Goransson, C. Black, y T. Culver, *Software defined networks a comprehensive approach*, Cambridge, MA: Elsevier, 2014.

[9] Q. Duan y M. Toy, Virtualized software-defined networks and services, Norwood, MA: Artech House, 2017.

[10] "SDN architecture, issue 1.1," ONF, Palo Alto, CA, ONF TR-521, 2016.

[11] P. Arlotto, "IT as a strategic resource," *Trustee*, vol. 67, pp. 37–38, marzo, 2014.

[12] J. Varia y S. Mathew, "Overview of Amazon Web Services," AWS, Seattle, WA, 2020. [Online]. Disponible: https://docs.aws.amazon.com/whitepapers/latest/aws-overview/aws-overview.pdf#document-details

[13] "Academy cloud foundations (ACF): module 01 student guide, v. 1.0.5, ," AWS, Seattle, WA, 100-ACFNDS-10-EN-SG, 2018. [Online]. Disponible: https://usermanual.wiki/Document/Module20120Student20Guide.1272554044.pdf

[14] J. Hurwitz, R. Bott, M. Kaufman, y F. Halper, *Cloud services for dummies*, Hoboken, NJ: WIley, 2014.

[15] G. Kim, K. Behr, y G. Spafford, "The Phoenix project: a novel about IT, DevOps, and helping your business win, Portland, OR: IT Revolution, 2013.

[16] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. Martin, S. Mellor, K. Schwaber, J. Sutherland, y D. Thomas, Manifiesto para el desarrollo ágil de software. [Online]. Disponible: https://agilemanifesto.org/iso/es/manifesto.html

[17] M. Shahin, M. Ali Babar, y L. Zhu, "Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017.

[18] M. Benítez. (2012). Análisis y mejora de los procedimientos de una empresa de ingeniería eléctrica. [Online]. Disponible: http://bibing.us.es/proyectos/abreproy/30176/

[19] S4N. (2019). ADN-S4N. [Online]. Disponible: http://s4n.co/es-que-es-s4n.html

[20] J. M. Cortés, *Cultura y procesos: un enfoque a la adopción de DevOps en S4N*, Bogotá, Colombia: S4N, 2017. [Online]. Disponible: https://docplayer.es/152850219-Cultura-y-procesos-un-enfoque-a-la-adopcion-de-devops-en-s4n.html

[21] E. Senabre-Hidalgo, "Adapting the Scrum framework for agile project management in science: case study of a distributed research initiative," *Heliyon*, vol. 5, no. 3, e01447, 2019.

[22] IBM Garage Field Guide, IBM, Armonk, NY, USA, 2019. [Online]. Disponible: https://www.ibm.com/cloud/architecture/files/ibm-garage-field-guide.pdf

[23] Amazon Web Service. (2018). Elección de una plataforma en la nube [Online]. Disponible: https://aws.amazon.com/es/choosing-a-cloud-platform/

[24] J. F. Gómez y Á. Pachón. (Jul. 2018). Definición infraestructura AWS– Ipinnovatech [Online]. Disponible: http://bit.ly/2OS3pSl

[25] J. F. Gómez, "Modelo operativo para el desarrollo y aprovisionamiento ágil de aplicaciones y servicios de TI," tesis de maestría, Cali, Colombia, Fac. Ingeniería, Universidad Icesi, 2019

[26] Amazon Web Service. (2019). AWS Trusted Advisor [Online]. Disponible: https://aws.amazon.com/es/premiumsupport/technology/trusted-advisor/

[27] J. F. Gómez. (Mar, 2019). Diagnóstico Ipanovatech (oportunidades mejora) [Online]. Disponible: http://bit.ly/384AnWV

[28] Amazon Web Service. (2019). AWS Config [Online]. Disponible: https://aws.amazon.com/es/config/

[29] Amazon Web Service. (2019). Instancias optimizadas para Amazon EBS [Online]. Disponible: https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/EBSOptimized.html

[30] Amazon Web Service. (2019). Despliegues multi-AZ de Amazon RDS [Online]. Disponible: https://aws.amazon.com/es/rds/details/multi-az/.

[31] Open Daylight Project. (2018). Open Day Light platform overview [Online]. Disponible: https://www.opendaylight.org/what-we-do/odl-platform-overview

[32] Amazon Web Service. (2019). System manager [Online]. Disponible: https://aws.amazon.com/es/systems-manager/

[33] *AWS systems manager: guía del usuario*, AWS, Seattle, WA, 2020. [Online]. Disponible: https://docs.aws.amazon.com/es_es/systems-manager/latest/userguide/sysman-manual-agent-install.html#agent-install-rhel

[34] Amazon Web Service. (2019). System manager features [Online]. Disponible: https://aws.amazon.com/es/systems-manager/features/

[35] J. F. Gómez, "GuíasLaboratorio." (Ene 17, 2020). Distribuido por GitHub. Disponible: http://bit.ly/2qbdvDP

[36] F. Méndez, *Descripción de procesos y mapa de procesos*, San José, Costa Rica, INEC, 2015. [Online]. Disponible: https://www.inec.cr/sites/default/files/documentos/inec_institucional/transparencia/procesos_institucionales/descripcion_del_mapa_de_procesos.pdf

[37] J. Beltrán, M. Carmona, R. Carrasco, M. Rivas, y F. Tejedor, *Guía para una gestión basada en procesos*, Andalucía, España: Instituto Andaluz de Tecnología, 2016. [Online]. Disponible: http://www.centrosdeexcelencia.com/wp-content/uploads/2016/09/guiagestionprocesos.pdf

[38] D. Farrow y J. Baker, "ISO 9001:2015. Elaboración de mapas de procesos," NHK, vol. 151, pp. 10–17, 2015.

[39] *Sistemas de gestión de la calidad*, ISO 9001:2015, International Standarization Organization, Ginebra, Suiza, sep. 2015.

[40] M. Gërvalla, N. Preniqi, y P. Kopacek, "IT infrastructure library (ITIL) framework approach to IT governance," *IFAC-Papers Online*, vol. 51, no. 30, pp. 181–185, 2018.

[41] T. Fontalvo Herrera, R. Quejada, y J. Puello Payares, "La gestión del conocimiento y los procesos de mejoramiento," *Dimens. Empres.*, vol. 9, no. 1, pp. 80–87, 2011.

[42] A. Canals-Parera, "La gestión del conocimiento en las organizaciones," *Novática*, vol. 196, pp. 57–60, 2008.

[43] M. Rouse, "Information security management system (ISMS)," 2011. [Online]. Available: https://whatis.techtarget.com/definition/information-security-management-system-ISMS.

[44] *Sistemas de gestión de la calidad – fundamentos y vocabulario*, ISO 9000:2005, International Standarization Organization, Ginebra, Suiza, 2005.

[45] N. Chapaval. (2017). Qué es frontend y backend. [Online]. Disponible: https://platzi.com/blog/que-es-frontend-y-backend/

[46] A. Menzinsky, G. López, y J. Palacio, *Scrum manager*, v. 2.6. Lubaris Info 4 Media SL., Zaragoza, España, 2016. Disponible: https://www.scrummanager.net/files/sm_proyecto.pdf

[47] M. Cohn, *User stories applied: for agile software development*, Boston, MA: Addison Wesley, 2012.

[48] J. I. Herranz. (2011). TDD como metodología de diseño de software. [Online]. Disponible: https://www.paradigmadigital.com/dev/tdd-como-metodologia-de-diseno-de-software/

[49] R. McLLree. (Ago. 27, 2007). How 'percent-complete' is that task again?. [Online]. Available: https://pmhut.com/how-percent-complete-is-that-task-again

[50] D. Zacarias. (2018). 20 product prioritization techniques: a map and guided tour. [Online]. https://foldingburritos.com/product-prioritization-techniques/

[51] M. Cohn, "Agile estimating and planning," *HortScience*, vol. 47, no. 12. pp. 1832–1836, 2012.

[52] J. Sandobalin, E. Insfran, y S. Abrahao, "Automatización del aprovisionamiento de infraestructura en la nube," en *Actas de las XIII Jornadas de Ingeniería de Ciencia e Ingeniería de Servicios (JCIS 2017)*, G. Ortiz, sept. 2017, sesión 5, ponencia 3.

[53] S. Chacon y B. Straub, *Pro Git*, Mountain View, CA: APress, 2014. [Online]. Disponible: https://git-scm.com/book/en/v2

[54] T. Hathaway y A. Hathaway, *How to write effective requirements for IT - simply put!: use four simple rules to improve the quality of your IT requirements*, Land O'Lakes, FL, BA Experts, 2016. [eBook Kindle].

[55] Trello Inc. Acerca de Trello [Online]. Disponible: https://trello.com/es/about

[56] Trello Inc. (2017). ¿Qué es Trello? [Online]. Disponible: https://help.trello.com/article/708-what-is-trello

[57] D. J. Anderson. (2014). Kanban an alternative path to agility presenter the meaning of agile [diapositivas].

[58] Slack Techonologies. (2019). Slack [Online]. Disponible: https://slack.com/intl/es-co/

[59] Slack Techonologies. (2019). ¿Qué es Slack? [Online]. Disponible: https://slack.com/intl/es-co/help/articles/115004071768-what-is-slack-

[60] J. F. Gómez, "Templates-AWS-IaC Watch." (oct. 13, 2019). Distribuido por Git Hub. Disponible: http://bit.ly/33w2QlX

[61] Amazon Web Services. (2019). AWS CodeBuild [Online]. Disponible: https://aws.amazon.com/es/codebuild/

[62] Amazon Web Services. (2019). AWS CodeDeploy [Online]. Disponible: https://aws.amazon.com/es/codedeploy/

[63] Amazon Web Services. (2019). AWS CodePipeline [Online]. Disponible: https://aws.amazon.com/es/codepipeline/

[64] The Jenkins project. (2019). What is Jenkins [Online]. Disponible: https://jenkins.io/doc/

[65] N. Swaraj. AWS Automation Cookbook. Birmingham, UK: Packt Publishing, 2017

Sistemas basados en RNA para clasificar malware en Android

Sebastián Felipe Landínez García, MSc.
Andrés Navarro Cadavid, Ph.D

Citación

S. F. Landínez y A. Navarro, "Sistemas basados en RNA para clasificar malware en Android," en *Aprovisionamiento ágil – Clasificación de malware – Optimización Giraph* [Bitácoras de la maestría, vol. 7], Cali, Colombia: Universidad Icesi, 2020, pp. 117-165.

RESUMEN

Los teléfonos inteligentes son dispositivos móviles con capacidad de procesamiento, esta les permite, además de realizar las funciones típicas de comunicación (*e.g.*, llamadas y mensajes de texto), conectarse a Internet, utilizar una cámara para fotos y vídeo, almacenar archivos, etc. Como cualquier sistema informático, están expuestos a amenazas como *malware*, robo de información y fraude electrónico. A pesar de que grandes compañías de seguridad ofrecen su servicio como aplicaciones, la literatura indica que tienen tasas de detección por debajo del 75 % bajo ciertas condiciones, y que los atacantes disponen de técnicas que les permiten evadir herramientas de detección tradicionales, como el método de firmas. Es necesario entonces explorar nuevas estrategias que mejoren la tasa de detección de *malware*. El presente estudio utiliza aprendizaje de máquina para abordar este problema, un enfoque que considera los patrones de la aplicación, en vez de características que pueden ser fácilmente modificadas, como su firma. En este documento se describe el diseño, implementación y evaluación de un sistema de aprendizaje de máquina basado en redes neuronales artificiales para la clasificación de *malware* en Android, capaz de clasificar un ejemplo como malicioso o no, a partir del aprendizaje de algunas características de aplicaciones previamente etiquetadas como *malware* o benévolas. Con este sistema se espera reducir las tasas de infección y los perjuicios derivados de ellas.

INTRODUCCIÓN

SITUACIÓN ACTUAL Y ANTECEDENTES

En la última década, los avances tecnológicos en: los componentes de los teléfonos inteligentes –RAM, dispositivos de almacenamiento externo e interno y cámara, entre otros–; la velocidad, confiabilidad y cobertura de las redes celulares que les proveen conectividad ubicua –LTE-A ofrece un pico teórico de 3 Gbps en enlace de bajada y 1.5 Gbs en enlace de subida [1] y se espera que 5G ofrezca hasta 20 Gbps de capacidad de descarga [2]–; los métodos de entrada; y la cantidad de aplicaciones existentes –redes sociales, comercio electrónico, banca electrónica, juegos, etc.– han popularizado estos dispositivos, al punto de ser casi una necesidad diaria.

Android es el sistema operativo de Google para teléfonos móviles, está basado en el kernel de Linux. Desde su presentación inicial en 2008 se ha ido consolidando como el sistema operativo más usado en dispositivos móviles, su porción del mercado a junio de 2020 es de 85,4 % [3]. El número de aplicaciones disponibles, si bien varía dependiendo de la fuente que se consulte, ronda los tres millones. Aunque fue originalmente desarrollado para teléfonos inteligentes, hoy es común encontrarlo en otros dispositivos, tales como: tabletas, televisores, dispositivos "usables" (*e.g.*, gafas y relojes de pulso) y vehículos [4]. La popularidad, portabilidad y capacidad de ejecutar tareas útiles y cotidianas en múltiples campos de la vida humana actual han provocado que progresivamente se confíe información más sensible a ellos, lo que los convierte en un blanco muy deseable para los cibercriminales.

Un reporte de McAfee sobre cibercrimen indica que el costo de su actividad ronda los 600 billones de dólares, esto es alrededor del 0.8 % del PIB mundial [5]. Menciona además que encriptar la información de un dispositivo y después pedir rescate por ella (*ransomware*) es la herramienta de cibercrimen con mayor crecimiento [5], [6] y [7].

Para los dispositivos móviles en particular se reportan múltiples tipos de amenazas. Felt et al. [8], después de una revisión muy extensa de *malware* móvil [8], destacan cambios importantes en las motivaciones, indican que aunque los primeros *malware* móviles tenían como único propósito molestar, los ataques que tienen como propósito el robo de información y el fraude económico se vienen incrementando. Los tipos más populares incluyen la venta de información

del usuario, el robo de sus credenciales, la su*scrip*ción indeseada a servicios *premium* y la solicitud de un rescate por la información. Dicen Felt et al que las principales herramientas de seguridad disponibles son la API (*Application Programming Interface*) de permisos y la revisión de aplicaciones que la tienda oficial realiza.

Una revisión más reciente [9] describe *malware* más complejo y nuevas técnicas para evitar la detección, como *data obfuscation*. Zhou y Jiang [10] realizaron una prueba con "cuatro software de seguridad móvil representativos", la que arrojó como resultado una tasa de detección de 79.6 % en el mejor caso y de 20.2 % en el peor.

Estos hechos exponen la necesidad de usar nuevos enfoques para detectar *malware* en Android. Los enfoques tradiciones incluyen el método de firmas, en el que una aplicación es representada por un ID único y se etiqueta como peligrosa, lo que permite que el software pueda alertar acerca de ella; sin embargo, técnicas como *data obfuscation* y reempaquetamiento superan estos mecanismos de seguridad.

Como respuesta, se han propuesto sistemas basados en aprendizaje de máquina, que identifican al *malware* por sus patrones y no mediante firmas: Arp et al. [11] usan el algoritmo de *Support Vector Machine* (SVM) directamente en el celular y el algoritmo de K-means para analizar los permisos de una aplicación; Wu et al. [12] las llamadas de la API; mientras Teufl et al. [13] analizan los metadatos de una aplicación mediante un algoritmo de aprendizaje de máquina (*Machine Learning*, ML).

ML es un subdominio de la inteligencia artificial (*Artificial Intelligence*, AI), la misma que está definida como la habilidad de un computador para aprender sin ser explícitamente programado [14]; con ella, en vez de construir un programa personalizado para cada problema, el computador sólo necesita aprender a comportarse respecto del problema, a través de un proceso llamado entrenamiento.

A pesar de que el concepto fue propuesto por Samuel a fines de los años 50 [14], no prosperó debido a la falta de métodos, técnicas y capacidad de procesamiento, y a los fracasos para obtener aplicaciones prácticas pese a las altas expectativas que se tenían en el momento sobre la investigación en este campo. En los últimos años, los métodos de aprendizaje de máquina están siendo aplicados cada vez con mayor frecuencia en diferentes campos, gracias

a la mayor potencia en los procesadores, la invención de nuevas técnicas y métodos para reducir el tiempo dedicado a los cálculos y el nuevo *hardware,* que se desempeña mejor en estas tareas. El aprendizaje de máquina, y en particular, las redes neuronales artificiales, ha sido utilizado en aplicaciones como electrónica de alta potencia [15], [16] y física de fluidos [17].

LA PROPUESTA

Lo dicho permite concluir que es posible aplicar técnicas de aprendizaje de máquina a la clasificación del *malware* en Android y que las tasas de detección usando estos enfoques son muy superiores a las tradicionales (94% [9] y 79.6 % [10], respectivamente). También que, a pesar de que muchos algoritmos han sido probados, no todos han probado múltiples hipótesis con diferentes grados de complejidad en sus algoritmos para hallar la que dé mejor ajuste; y que ninguno de estos sistemas, según nuestro conocimiento, fue diseñado como un *script* para ser utilizado como un componente en un sistema más grande.

Por todo ello, para este proyecto se propuso un sistema basado en redes neuronales artificiales, diseñado como un *script* que pueda ser usado como parte de desarrollos futuros, un sistema dinámico que permita seleccionar arbitrariamente el número de capas y el número de neuronas por capa con el propósito de detectar *malware* en Android, de manera que se pueda ofrecer una nueva técnica de detección que se desempeñe mejor que las técnicas estándar existentes.

El principal problema que este sistema intenta atacar son las bajas tasas de detección de *malware* con los métodos tradicionales, las que influencian directamente a las tasas de infección, los costos asociados y la confianza de los usuarios en sus teléfonos inteligentes y sus aplicaciones. Para mejorar las tasas de detección se debe investigar nuevas estrategias sensibles a herramientas avanzadas de ataque, como *data obfuscation*.

Se han desarrollado y probado sistemas de aprendizaje de máquina que presentan resultados superiores comparados con los enfoques tradicionales, con varios tipos de algoritmos probados, incluyendo las redes neuronales.

Excepto Art et al. [11], quienes realizan el análisis en el celular, los sistemas mencionados realizan su análisis en un servidor, por tanto: no es posible notificar al celular en el momento de instalación o se puede sobrecargar el celular por el procesamiento adicional que implica realizar estos cálculos. Se

puede notar que se han encontrado pocos sistemas para esta aplicación en particular que hayan probado varios valores de hiperparámetros con el fin de hallar los más óptimos, ni tampoco un sistema diseñado con el fin de funcionar a la vez como un sistema monolítico o como parte de un sistema más grande.

Aunque Android ha desarrollado nuevas estrategias de seguridad, como los permisos en tiempos de ejecución, esta mejora trabaja únicamente a partir de la versión 6.0, por lo tanto, las versiones previas aún son vulnerables. Además, los usuarios no experimentados que no están familiarizados con los permisos y su significado pueden instalar *malware* en sus celulares sin darse cuenta de ello.

El objetivo general del proyecto es desarrollar un sistema de aprendizaje de máquina basado en redes neuronales artificiales para detectar *malware* en Android, y sus objetivos específicos: diseñar e implementar el sistema basado en redes neuronales artificiales; analizar los hiperparámetros para encontrar los que otorguen mejores resultados; y evaluar el sistema y comparar los resultados con otras estrategias de detección tradicional y algunas basadas en ML.

Entre los retos se destaca que, como cualquier sistema de aprendizaje de máquina, el presente necesitará un *dataset* de un tamaño considerable para obtener *datasets* de entrenamiento, validación y prueba a través de un proceso de *hold-out* o validación cruzada [18], sin embargo, no hay muchos *datasets* de *malware* disponibles

Por otra parte, los propietarios de los *dataset* de *malware* disponibles son universidades e investigadores, ellos proveen acceso después de algún tipo de verificación (debido a consideraciones éticas y el uso destructivo que se le puede dar a estos ejemplos).

También, hay una falta detectable de *datasets* que estén probados para ser libres de *malware*, lo que puede llevar a sesgo en el entrenamiento y la clasificación, porque un grupo de aplicaciones etiquetadas como benéficas pueden resultar ser *malware*.

El proceso propuesto para este estudio es: primero, el sistema debe ser entrenado para aprender qué patrones tienen las aplicaciones perjudiciales y cuáles las benévolas; a continuación, el sistema se prueba con ejemplos que no ha visto aún, para medir sus habilidades de generalización; después, se prueban múltiples arquitecturas para encontrar cuales son los parámetros óptimos de configuración de la red neuronal; y finalmente, se compara el desempeño del sistema con el de sistemas similares.

MARCO TEÓRICO

APRENDIZAJE DE MÁQUINA

Este campo de la inteligencia artificial estudia las técnicas que permiten a un computador desempeñarse progresivamente bien, esto es aprender de una tarea sin ser explícitamente programada para ella.

Las primeras teorías acerca de máquinas pensantes fueron formuladas por Alan Turing [19], mediante ideas como la máquina y el test de Turing. Posteriormente, se crearon los primeros algoritmos de AI, entre ellos las redes neuronales y los perceptrones [20]. Sin embargo, hubo un descenso en los esfuerzos en investigación en este campo debido a la falta de resultados prácticos.

No fue sino hasta el siglo XXI, cuando: el incremento en la capacidad de cálculo de los procesadores y en la capacidad de almacenamiento de los computadores; la democratización en el acceso de los computadores –un computador de gama media accesible para mayoría de las personas puede ejecutar aceptablemente algunas tareas de *machine learning*–; las nuevas técnicas para acelerar los cálculos, como la aceleración por GPU (*Graphics Processing Unit*); un volumen incrementado de información que puede ser recolectada y transferida debido al auge de Internet y la enorme cantidad de datos que pueden ser obtenidos con los nuevos tipos de sensores, tales como cámaras, sensores de movimiento y calor, pantallas táctiles y acelerómetros; y las nuevas técnicas de aprendizaje de máquina, como el *deep learning*, hicieron posible el resurgir de este campo, el cual está siendo usado para un gran número de aplicaciones, entre ellas: reconocimiento de dígitos escritos a mano [21]; visión de máquina y reconocimiento de patrones [22]; procesamiento de lenguaje natural (*e.g.*, creación automática de subtítulos para videos, traducción, análisis de sentimientos) [23]; y recomendación de películas y series con base en los gustos de usuarios similares [24]. Justo ahora, el *machine learning* se usa para un número creciente de aplicaciones, y debido a que los factores mencionados tienden a seguir aumentando, es factible que sea una de las áreas más prominentes de la computación.

Algoritmos de aprendizaje de máquina

Los algoritmos de aprendizaje de máquina se clasifican como: de aprendizaje no supervisado, de aprendizaje supervisado y de aprendizaje reforzado.

Los algoritmos de aprendizaje no supervisado tratan de extraer alguna estructura de datos que no han sido etiquetados, es decir, busca patrones o rasgos comunes en los datos que le son suministrados. Pueden ser usados para tareas como *clustering*, detección de anomalías, reducción de dimensionalidad, etc. Algunos algoritmos de aprendizaje no supervisado son: K-means, mapas auto-organizados o redes de Kohonen [25], clustering jerárquico y clustering espacial basado en densidad (DBSCAN) y factor atípico local

El aprendizaje reforzado es un área del aprendizaje automático que tiene como inspiración la psicología conductista, ella se refiere a la capacidad de determinar qué acciones deben ser ejecutadas por un software con el propósito de maximizar una recompensa acumulativa. La esencia del proceso es que el software pueda seleccionar y ejecutar la mejor alternativa, eligiéndola a través de un refuerzo positivo. El modelo de aprendizaje reforzado posee cinco elementos, que corresponden a un proceso estocástico denominado problema de decisión de Markov (MDP, *Markov Decision Process*): pasos para la decisión, conjunto de estados posibles, conjunto de acciones posibles, reglas de transición y reglas de determinación de las recompensas. Entre las aplicaciones de algoritmos de este tipo se encuentran: Mobileye, un sistema avanzado de asistencia a la conducción basado en la visión, que advierte al conductor para evitar posibles colisiones; y Q-Learning, que fue inventado hace más de 25 años y está basado en MDP [26].

Por su parte, el aprendizaje supervisado utiliza datos que han sido etiquetados *a-priori*, así puede predecir características de un nuevo ejemplo basado en las características de ejemplos similares; pueden ser usados para tareas como clasificación y regresión. Su proceso típico es:

- escoger cuáles características de los datos serán consideradas para las tareas de predicción.
- dividir aleatoriamente el *dataset* obtenido en *datasets* de entrenamiento, validación y prueba.
- dependiendo del algoritmo escogido, definir una función que transforme las características de la entrada en características de salida e indique qué tan diferentes son las etiquetas asignadas de las etiquetas correctas;
- entrenar el algoritmo suministrándole el *dataset* de entrenamiento. Esto significa, reducir el valor de la función costo al mínimo usando técnicas como el descenso de gradiente; y

- cuando los parámetros están ajustados de tal modo que la función de costo está en su mínimo, usarlos para predecir etiquetas para el *dataset* de prueba que aún no ha visto el algoritmo

Su desempeño puede ser reflejado con estadísticas como la precisión, la sensibilidad y el F-score, entre otras. Algunos ejemplos de algoritmos supervisados son: la regresión lineal, la regresión logística, las SVM, los modelos Naive Bayes, los árboles de decisión y las redes neuronales.

Descenso de gradiente

Esta es una técnica utilizada para hallar mínimos en funciones multivariable basada en el concepto de gradiente, el mismo que se define según (1), donde ∂ representa una derivada parcial, es decir, en una única dirección, y el gradiente cumple con la propiedad de, al ser evaluado en cualquier punto, siempre apuntar en la dirección de máximo crecimiento ($-\nabla f$). Aplicando

$$F(x_1, x_2, \ldots, x_n) = \left[\frac{\partial F}{\partial x_1}, \frac{\partial F}{\partial x_2}, \ldots, \frac{\partial F}{\partial x_n} \right] \quad (1)$$

este concepto, es posible dirigirse hacia el mínimo moviéndose en la dirección del gradiente, lo que puede realizarse de acuerdo con (2), en donde η es un hiperparámetro llamado tasa de aprendizaje, el cual indica cuánto se debe recorrer en la dirección contraria a la del gradiente.

$$\Delta x = -\eta \quad F(x) \quad (2)$$

Una tasa de aprendizaje baja puede hacer que el algoritmo se tarde mucho en encontrar ese mínimo, pero una tasa de aprendizaje alta también puede ser contraproducente. Con esta técnica es posible converger a un mínimo local sin necesidad de hallar segundas derivadas, lo que implicaría cálculos adicionales.

Redes neuronales artificiales

Hasta ahora el cerebro humano, compuesto por aproximadamente 86 billones de neuronas, es la mejor "máquina" conocida para resolver problemas. Una

neurona es una única unidad que tiene un número de entradas –las dendritas–, y un elemento de salida –el axón–, capaz de conectarse a dendritas que pertenecen a otras neuronas. La intensidad de la señal en las conexiones entre neuronas –sinapsis–, puede modificarse; se cree que el cerebro aprende al alterar dichos pesos de acuerdo con los estímulos externos. Una Red Neuronal Artificial (RNA) es una técnica que trata de emular el comportamiento del cerebro humano, compuesto por múltiples unidades interconectadas entre ellas, llamadas neuronas, en donde cada una realiza un pequeño cálculo y pasa el resultado como entrada a la siguiente capa de neuronas.

Las RNA son un algoritmo de aprendizaje supervisado que se inventó en la década de los 50. Él trata de emular el comportamiento del cerebro humano al interconectar una o múltiples capas de unidades llamadas neuronas. Igual que en el cerebro, las neuronas se interconectan entre sí y la conexión entre dos de ellas tiene un peso que indica cuánto impacto tiene una sobre la otra; los pesos de las interconexiones entre las neuronas son los parámetros de las RNA que pueden ser alterados. Al cambiar los pesos, una RNA puede aprender patrones en los datos y realizar tareas –como clasificarlos según unas etiquetas predefinidas o predecir un valor a partir de ejemplos de entrada (regresión)–.

Una neurona toma los pesos ponderados de la capa anterior y, usando una función de activación, transfiere a su salida a las neuronas conectadas a ella. Existen varias funciones que pueden utilizarse como funciones de activación, como por ejemplo:

- la función de Heaviside (3) o función escalón;

$$f(x) = \begin{cases} 0 & si\ x \leq 0 \\ 1 & si\ x > 0 \end{cases} \tag{3}$$

- la función softmax (4) o función exponencial normalizada, que toma un vector K-dimensional z con valores reales arbitrarios y devuelve un vector K-dimensional con valores en el rango [0, 1];

$$\theta(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_j}} \text{ para } j = 1, \ldots, k \tag{4}$$

- La función tangente hiperbólica (5).

$$tanh(x) = \frac{senh(x)}{cosh(x)} \qquad (5)$$

- La función rectificador, definida como (6)

$$f(x) = max\ (0,x) \qquad (6)$$

Para el sistema desarrollado, que trata un problema de clasificación, se utilizará como función de activación la función sigmoide (7). Su rango está entre 0 a 1: para grandes valores negativos de entrada, su salida se acerca a 0; y para grandes valores positivos de entrada, su salida se acerca a 1. Su característica más valiosa es su comportamiento en la zona central, donde un pequeño cambio en la entrada causa un pequeño cambio en la salida. De esta manera, es posible ajustar suavemente los pesos para obtener la salida deseada.

$$g(x) = \frac{1}{1+e^{-x}} \qquad (7)$$

Otro de los elementos que define el comportamiento de una red neuronal es el tipo de sinapsis que existe entre las neuronas de cada capa. Este proceso puede ser realizado de múltiples maneras, por ejemplo, mediante redes: neuronales convolucionales, neuronales recurrentes y perceptrón multicapa.

Las redes neuronales convolucionales están basadas en el funcionamiento de las células en la corteza visual [27], ellas se caracterizan por responder únicamente a estímulos que provienen de una región restringida del campo visual, conocida como campo receptivo y son muy usadas para reconocimiento de video e imágenes [28], [29], sistemas de recomendación [30] y procesamiento de lenguaje natural [31].

Una red neuronal tradicional asume que todas las entradas (elementos del *dataset*) son independientes unas de las otras, pero para algunas tareas, como por ejemplo el procesamiento de lenguaje natural –tratar de predecir qué palabra

vendrá a continuación–, es necesario considerar cuáles eran los estados previos de la red [32] [33]. Las redes neuronales recurrentes tienen una "memoria" en la que pueden guardar sus estados previos y así permitirles influir en las siguientes predicciones.

En las redes completamente conectadas –que es como se conoce a las perceptrón multicapa–, las neuronas de una capa son afectadas por todas las neuronas de la capa previa, contrario a las redes neuronales convolucionales. Consideran más características por neurona, lo que puede hacerlas mucho más complejas. Por ejemplo, una red neuronal completamente conectada que considere una imagen de 100x100 pixeles (10.000 características de entrada) tendría 10.000 pesos en la segunda capa por cada neurona en ella.

Perceptrón multicapa

Como se mencionó en el párrafo anterior, un perceptrón multicapa es una red neuronal en la cual cada neurona está conectada a todas las demás en la siguiente capa, lo que puede crear una gran cantidad de pesos, pero como ventaja, cada entrada afectará a la salida obtenida. Los parámetros de una red neuronal son matrices que contienen los pesos de las conexiones entre una neurona y las neuronas en la siguiente capa, representados como $\theta_{i,j}^{(l)}$, donde i y j son las neuronas interconectadas y l es la capa en cuestión.

Estos valores serán alterados con el fin de hallar el menor valor de la función de costo. Los hiperparámetros de la red son valores que están relacionados a su arquitectura y que no serán modificados por el algoritmo que busca la mínima función de costo, estos son: el número de capas consideradas, el número de neuronas en cada capa y algunos parámetros adicionales relacionados con técnicas implementadas para obtener mejores resultados. La primera y última capa de una red neuronal artificial son llamadas las capas de entrada y salida. La capa de entrada tiene un número de neuronas que equivale al número de características que se consideran para cada ejemplo, si la red neuronal se destinará a una tarea de clasificación, la capa de salida tendría un número de neuronas igual a cuantas etiquetas o clases tiene el problema, salvo el caso de que hayan sólo dos clases de salida, en el cual sólo se necesitaría una neurona (clasificación binaria). A las capas que hay entre las capas de entrada y salida se les llama capas ocultas. En el dominio de las redes neuronales hay un área denominada *deep learning*, en ella las redes neuronales tienen más de tres capas, esto es, más de una capa oculta [34].

En el caso de una red neuronal completamente conectada que aborda un problema de clasificación, la forma más utilizada para obtener etiquetas de salida a partir de características de entrada es el algoritmo de programación hacia el frente. La salida de la red se calcula capa por capa, y el valor de cada capa se calcula usando exclusivamente las salidas de la capa previa (8), donde: $\Theta(n)$ son los parámetros para la capa n; $a(n)$ son las unidades de activación (entradas) de la capa n; $a(n+1)$ son las unidades de activación para la siguiente capa; y $g(x)$ es la función de activación, que puede ser una de las funciones mencionadas.

$$a^{(n+1)} = g(\theta^{(n)} \times a^{(n)} \tag{8}$$

De esta manera el proceso se repite hasta alcanzar la capa de salida. En ella los valores corresponderán a la probabilidad de que este ejemplo pertenezca a alguna clase de salida. Para obtener las etiquetas resultantes, se define un umbral: las etiquetadas con una probabilidad mayor al umbral son un 1, mientras que las que estén por debajo son un 0. La función de costo (9) para una red neuronal dedicada a la clasificación corresponde a un promedio del error por cada ejemplo, donde: m es el número de ejemplos en el *dataset*; K es el número de etiquetas de salida; $h\Theta(x(i))k$ corresponde a la función de activación en la capa de salida –para el ejemplo i y la etiqueta de salida k–, $y(i)$ es el valor para la etiqueta de salida k para el ejemplo i. La función de costo dará: un valor elevado, si la etiqueta real del ejemplo y la salida obtenida son diferentes; o un valor cercano a cero, si la salida obtenida es similar a la etiqueta real del ejemplo.

$$J(\theta) = -\frac{1}{N}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-h_\theta(x^{(i)}))_k\right] \tag{9}$$

Overfitting y underfitting

Los algoritmos supervisados de aprendizaje de máquina están relacionados con una hipótesis que se expresa como una ecuación. Dependiendo del tipo de función, el número de términos y algunos hiperparámetros específicos a

cada algoritmo (*e.g.*, el número de vecinos considerados en el *K-nearest neighbors*, el número de capas, el número de neuronas por capa en las RNA, etc.), la hipótesis puede ser más o menos compleja.

Si la hipótesis no es suficientemente compleja, el sistema no va a ser capaz de determinar los patrones de la información; pero si la hipótesis es demasiado compleja, el sistema podría haber aprendido de memoria los patrones del *dataset* de prueba y por tanto sería un pobre predictor para cualquier otro *dataset*. Estos problemas se llaman *underfitting* y *overfitting*, respectivamente.

Para reducir el underfitting es posible alterar los hiperparámetros del algoritmo y añadir complejidad a la hipótesis (*e.g.*, en el caso de una red neuronal, añadir más neuronas o una capa adicional); por su parte, para mitigar el *overfitting*, se agrega aleatoriedad a la hipótesis a través de técnicas como añadir una unidad de sesgo –*bias unit*– en cada capa [35], es decir, una neurona cuya salida es siempre uno. Otra técnica utilizada para reducir el *overfitting* se llama regularización L2 [36] y consiste en agregar una suma cuadrática de los parámetros de la capa a la función de costo (9), según (10).

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-h_\theta(x^{(i)}))_k\right] + \frac{1}{2m}\sum_{l=1}^{L}\sum_{i=1}^{sl}\sum_{j=1}^{sl+1}\left(\theta_{j=1}^{(l)}\right)^2 \quad (10)$$

λ es llamado un parámetro de regularización que indica cuán importante debe ser el término de regularización L2 y se convierte en otro hiperparámetro de la red. Esta función añade un gran valor a la función de costo si los parámetros son altos, de manera que castiga las hipótesis complejas. También, es necesario remarcar que el incremento debido a esta función es exponencial.

Algoritmo de propagación hacia atrás

Después de calcular la función de costo es necesario usar un algoritmo para hallar su gradiente y minimizarlo. El método más usado esta tarea es el de propagación hacia atrás [37], él está basado en la idea de calcular el error en la capa de salida y, entonces, propagarlo hacia atrás a las capas previas, hasta que se llega a la capa de entrada.

El error o desviación de la primera capa (δ) se calcula restando el valor de la unidad de activación en la capa de salida $(h(i)(x))$ menos la etiqueta real del ejemplo $(y(i))$; a continuación, el valor de la desviación de la capa previa se halla usando (11), donde gt es la derivada de la función de activación.

$$\delta^{(n)} = (\theta^{(n)})^{T\delta(n+1)} \times g'(a^{(n-1)}) \times \theta^{(n-1)} \quad (11)$$

Suponiendo que la función de activación sea la función sigmoide, gt corresponde a (12)

$$g'(x) = g(x)(1 - g(x)) \quad (12)$$

Este cálculo se debe realizar para cada ejemplo en el *dataset*, hasta la capa de entrada. La derivada de la función de costo respecto de los parámetros de la capa $\Theta(l)$ serán finalmente (13):

$$\frac{\partial}{\partial\theta^{(l)}} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} a_{(j)}^{(l)} \delta_{(i)}^{(l+1)} + \lambda\theta_{(ij)}^{(l)} \quad (13)$$

Es importante notar que el término de regularización L2 también está incluido en el gradiente y, en consecuencia, afecta su cálculo.

Métricas de evaluación

Para evaluar cuán bien se desempeña un sistema de clasificación existen múltiples métricas de evaluación, entre ellas: exactitud, matriz de confusión, precisión, sensibilidad y puntaje F1.

La exactitud [*accuracy*] es la tasa que mide cuántas instancias fueron clasificadas correctamente por el sistema (14).

$$Accuracy = \frac{elementos\ clasificados\ correctamente}{Total\ elementos} \quad (14)$$

La matriz de confusión es una tabla donde se reúne información acerca de la clasificación y permite visualizar el desempeño de un algoritmo; confronta los elementos de la clase predicha contra las verdaderas clases de cada elemento (Figura 1).

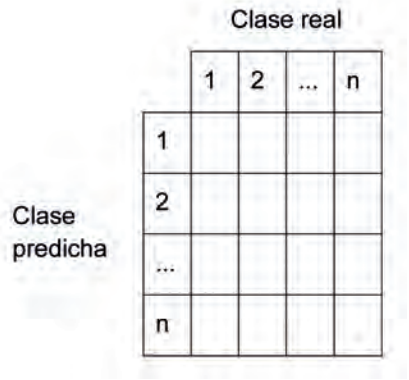


Figura 1. Matriz de confusión

Si el problema de decisión es binario, la matriz de confusión se reduce (Figura 2); esta distribución permite agregar nuevas métricas para los sistemas de clasificación.

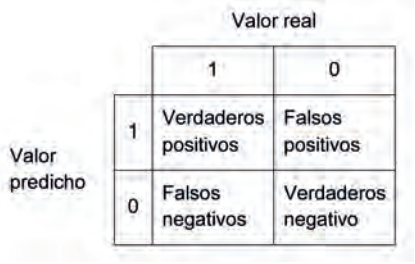


Figura 2. Matriz de confusión (problema binario)

La precisión (15), también llamada valor positivo predicho, es la fracción que muestra cuántas instancias seleccionadas o calificadas como 1 son relevantes, mide cuántos falsos positivos obtuvo el sistema –a mayor valor de precisión, menor número de falsos positivos–.

$$Precisión = \frac{Verdaderos\ positivos}{Verdaderos\ positivos + Falsos\ positivos} \qquad (15)$$

La sensibilidad, exhaustividad o *recall* (16) es la fracción que indica cuántas de las instancias relevantes fueron recuperadas, mide cuántos falsos negativos obtuvo el sistema –a mayor sensibilidad, menor número de falsos negativos.

$$Recall = \frac{Verdaderos\ positivos}{Verdaderos\ positivos + Falsos\ negativos} \qquad (16)$$

La sensibilidad y la precisión son medidas de cómo se desempeño un sistema de clasificación respecto de una prueba; ambas cifras contienen información importante acerca de dónde el sistema no se desempeñó correctamente, sin embargo, un clasificador podría tener precisión perfecta y una sensibilidad no tan buena. Con el fin de agrupar ambos marcadores en un sólo número surge el puntaje F1 (17), el cual da información acerca de los dos valores.

$$F_1 = \frac{Precisión \times Recall}{Precisión + Recall} \qquad (17)$$

ANDROID

Android es un sistema operativo móvil basado en un kernel de Linux, fue diseñado considerando los requisitos particulares de un celular, como sus capacidades limitadas de procesamiento y almacenamiento y la importancia central del consumo de batería, entre otros. Esto añade algunas restricciones y consideraciones al programador, por ejemplo: el hilo de la interfaz gráfica no puede ser usado para operaciones bloqueantes o que consuman mucho tiempo; el sistema operativo puede destruir en cualquier momento cualquier componente que esté corriendo debido a falta de procesamiento, y así sucesivamente. De hecho, a diferencia de las aplicaciones de escritorio, que en la mayoría de casos corren como un sólo proceso monolítico, una aplicación de Android está construida con múltiples componentes, incluyendo: actividades, servicios, fragmentos, *content providers*, *broadcast receivers*, entre otros [38].

La mayoría de los componentes mencionados están definidos en un archivo XML obligatorio para cada aplicación, llamado el AndroidManifest. Además de la definición de los componentes de la aplicación, contiene los permisos que la aplicación solicita, las características *hardware* y software, etc. La estructura típica de un AndroidManifest se muestra en la Figura 3.

Figura 3. Muestra de un AndroidManifest

Las aplicaciones Android pueden programarse en Java o en Kotlin, un lenguaje de programación que puede ser ejecutado sobre una máquina virtual de Java. Parte integral de la pila de software es su versión de la máquina virtual de Java, la cual permite que corran las aplicaciones programadas en dichos lenguajes. Hasta la versión 4.4 (Kitkat), la máquina virtual utilizada era Dalvik, a partir de entonces, se cambió a Android Runtime (ART). El código escrito en Java se compila a *bytecode* para la máquina virtual, que a su vez se almacena en archivos .dex (*Dalvid Executable*) y .odex (*Optimized Dalvik Executable*).

Las aplicaciones de Android se empacan comúnmente como APK (*Android Application Package*), que encapsulan el AndroidManifest y otros recursos de la aplicación, como: el archivo classes.dex, que contiene los archivos .class de Java y las librerías .jar convertidas a *bytecode* de Dalvik –formato reconocido por la máquina virtual–; recursos como imágenes XML que definen la disposición de las interfaces de la aplicación; y los certificados que la identifican, entre otros.

Estrategias de seguridad

Por ser el sistema operativo móvil más usado a nivel global, la seguridad es una de las principales preocupaciones de sus desarrolladores. Por esta razón, se han creado múltiples estrategias para luchar contra el *malware* y proteger a sus

usuarios de infección, robo de información, estafa y posible daño. También, han diseñado su sistema de modo que un programador pueda construir sus aplicaciones con los permisos y sistemas existentes por defecto y eviten decisiones difíciles acerca de la seguridad [39].

Una de las estrategias de seguridad incorporadas en Android es el Android Application Sandbox [40] –similar a la protección basada en usuarios de Linux que permite identificar y aislar recursos–, en la que Android asigna un ID único (*Unique IDentifier*, UID) a cada aplicación y la corre en un proceso propio, lo que aísla a las aplicaciones entre sí y las protege –a ellas y al sistema operativo– de aplicaciones maliciosas.

A través de ese identificador único, Android crea un *application-sandbox* a nivel del kernel, por estar a ese nivel, todo el software que está soportado en él –librerías del sistema operativo, las variables de tiempo de ejecución–, y todas las aplicaciones corren en el *sandbox* y están protegidas por él.

Otras estrategias de seguridad incorporadas en Android son: las implementaciones robustas de criptografía, permisos y comunicación interprocesos (*Inter-Process Communications*, IPC) segura incluida con el *framework* de aplicaciones; un sistema de archivos encriptado, que puede ser habilitado para proteger la información en dispositivos perdidos o robados; y permisos otorgados por el usuario para restringir el acceso a características del sistema y la información.

El último ítem citado se conoce como la API de permisos y es la principal estrategia de seguridad implementada por Android,. Si cualquier aplicación de Android desea utilizar alguna característica del sistema, como acceso a Internet, lectura y escritura de SMS, leer el registro de llamadas, etc., debe solicitarlo explícitamente añadiendo una línea <uses-permission ... > en el Android Manifest.

Después de solicitarlo, el sistema operativo informará al usuario en pantalla que la aplicación está solicitando un permiso para alguna característica del sistema, y depende de él otorgarlo o no. Antes de la versión 6.0 de Android (Marshmallow), la confirmación de todos los permisos solicitados por una aplicación se mostraba antes de su instalación. Si el usuario no estaba de acuerdo con ellos, la aplicación no era instalada. A partir de esta versión, el usuario puede permitir algunos permisos y negar otros y solicitarlos en tiempo de ejecución.

Malware

Malware es la abreviatura de *malicious* software y está definido como software específicamente diseñado para irrumpir, dañar o ganar acceso no autorizado a un sistema computacional [41]. Aunque los primeros ejemplos de *malware* fueron creados con el fin de molestar, casi como una broma, su aparición demostró que este sistema operativo y sus procesos de seguridad presentan vulnerabilidades [4]. El auge de Internet permitió que el *malware* pudiera expandirse mucho más rápidamente e infectar a un número mayor de personas.

El primer gusano de Internet, el gusano Morris, explotaba una vulnerabilidad en el servicio de correo de Unix y lanzaba un proceso que intentaba adivinar las contraseñas de otros computadores. El principal problema era que los computadores podrían infectarse más de una vez, apilando múltiples procesos y haciendo que se detuviesen, como en un ataque de denegación de servicios. Se dice que este virus afectó a 6.000 de 60.000 computadores en total en la ARPANET, el precedente del Internet actual [42].

El *malware* progresivamente se hizo más y más dañino, debido a que sus creadores podían sacar ganancias de él. Nuevos tipos surgieron, entre ellos: *phishing*, pretender ser una entidad confiable con el fin de robar información; *ransomware*, encriptar la información de la víctima y pedir rescate por ella; y robo de sesiones, robar las credenciales de la víctima durante una transacción importante y hacerse pasar por ella con propósitos maliciosos). Tal como ocurre con el *malware* web y de escritorio, el *malware* en Android ha evolucionado.

Las transformaciones triviales de *layout* y la reflexión son algunas de las estrategias de evasión. En la primera, si una aplicación de *malware* es detectada a partir de su *checksum* (firma), es relativamente fácil descompilarla, alterar algunos archivos sin cambiarla profundamente o firmar la aplicación con diferentes llaves, de modo que la suma de chequeo es diferente y se evadan los chequeos relacionados a las firmas; la segunda puede ser usada para acceder a todas los métodos privados y ocultos de la librería de la API, convertir una llamada a un método en reflectiva hace difícil rastrearla y, si el nombre del método está encriptado, haría imposible analizarlo estadísticamente.

Análisis estático y análisis dinámico

El análisis estático examina un programa sin correr código de él. Todos los métodos estáticos son vulnerables a técnicas de ofuscación, como encriptación,

que pueden remover o limitar el acceso al código. También, la inyección de código no Java, la actividad de la red y la modificación de objetos en tiempo de ejecución están fuera del alcance del análisis estático porque se realizan durante la ejecución. Está basado en el análisis de los archivos de la aplicación o del código fuente, por ejemplo, aquellos que se obtienen cuando un paquete APK se descompila. Los contenidos del APK son los siguientes:

- el directorio META-INF, que contiene el archivo Manifest, los certificados RSA de la aplicación y una lista de los recursos y los digestores SHA-1 de las líneas correspondientes a los recursos en el Manifest;
- el directorio Assets, que incluye los archivos que la aplicación puede recuperar a través de la clase AssetManager;
- el AndroidManifest.xml, un archivo que contiene el nombre del paquete, los permisos, la versión de la aplicación, las librerías solicitadas por la aplicación y una lista de los componentes de la aplicación (*i.e.*, servicios, actividades, *broadcast receivers*, *content providers*, etc.);
- el archivo classes.dex, que contiene todas las clases de Android compiladas en un formato dex que es reconocible por la máquina virtual Dalvik (si el celular tiene ART, los archivos .odex que reconocía Dalvik son reemplazados por archivos de formato enlazable y ejecutable (ELF, *Executable and Linkable Format*);
- el directorio lib, que contiene código compilado en subdirectorios específicos a la capa de software del procesador y que se nombra según él (*e.g.*, x86, x86 64, arm64-v8a);
- el archivo res, que contiene recursos no compilados en resources.arsc; y
- el archivo resources.arsc, que contiene recursos precompilados.

De ellos, para el análisis estático y la detección de *malware*, los más importantes son: el AndroidManifest.xml y el classes.dex.

Otro posible objeto de estudio en el análisis estático son los *intents*. Un *intent* se utiliza para comunicar componentes de Android, incluso entre aplicaciones diferentes. Además, el Manifest debe contener el *hardware* solicitado por la aplicación. En estos archivos pueden haber algunos patrones extraños, como por ejemplo, una calculadora que solicite usar la red celular y GPS.

Es posible también examinar los archivos DEX, el *bytecode* compilado para Dalvik, como no son legibles para el ser humano, se necesita una herramienta

para descompilarlos en un formato más comprensible, como Soot [43]. En general, entre más drástico es el proceso de descompilación, mayor es la probabilidad de error, porque se pierde cierta información al convertir de un formato al otro. A partir de los archivos descompilados se puede extraer las características de las clases, los llamados a la API, y las secuencias de estructuras y grafos de dependencia del programa pueden ser extraídos y analizados para hallar patrones peligrosos en el código.

El descompilador es una de las herramientas más utilizadas en el análisis estático, funciona como una herramienta de ingeniería inversa que intenta crear código fuente de alto nivel a partir de un archivo ejecutable o algún tipo de aplicación empaquetada. En el caso de Android, los descompiladores toman un archivo en formato APK y devuelven los archivos que lo componen – como el classes.dex y el resources.arsc–, otros archivos XML que definen la disposición de las interfaces, los estilos, constantes de la aplicación, recursos como imágenes png y el AndroidManifest. Actualmente, existen múltiples herramientas para realizar la descompilación, como decompile-apk [44], APK decompiler app [45] y decompile-android [46].

El análisis dinámico, por su parte, ejecuta el programa y observa sus resultados. En su aplicación más sencilla, se provee acceso a únicamente un camino de ejecución simultáneamente; sin embargo, a través de simulaciones, pueden evaluarse más caminos de ejecución. Muchos comportamientos de la aplicación pueden ser inducidos a través de las interfaces, los *intents* recibidos o disparadores de eventos automáticos [47]. No obstante, al ejecutar *malware* se abre la posibilidad que la aplicación maliciosa vea e interactúe con su entorno. Debido a esto, se han postulado dos tipos de análisis dinámico: *in-the-box* y *out-of-the-box*.

En el análisis *in the box*, la evaluación ocurre al mismo nivel de privilegios que el *malware*. Esto generalmente requiere alterar el sistema operativo o la máquina virtual del dispositivo y tiene la ventaja de que es fácil acceder a herramientas como las estructuras de memoria y datos de alto nivel del SO y de que puede ser posible acceder a librerías, métodos y las API –pero esto depende de los permisos que se tengan–; la desventaja es que al correr en la misma capa que el *malware*, le deja abierta la posibilidad de atacar la herramienta de análisis o evadirla, de hecho, han habido reportes de *malware* que altera su comportamiento cuando detecta un entorno simulado [48]. Otras desventajas del análisis *in-the-box* son: la necesidad de una modificación diferente para cada

una de las versiones de Android, debido a que se altera el SO o la máquina virtual y que potencialmente pueden haber más *bugs*.

En los métodos de análisis *out of the box*, el *malware* se utiliza con herramientas de emulación y virtualización para proveer seguridad mejorada a través del aislamiento del *malware*. Las herramientas de emulación completa proveen, además de todas las funcionalidades del sistema, acceso a periféricos como CPU, memoria y, en el caso de los *smartphones*, cámara, GPS, acelerómetros y otras herramientas. Algunos emuladores de Android puro son Andrubis [49], DroidScope [50] y CopperDroid [51]. Están construidos sobre QEMU, un emulador de código abierto para *hardware* ARM. Con algunos avances recientes, comportamientos de alto nivel que se creían no reproducibles con análisis *out-of-the-box* –como la IPC–, han sido emulados, incluso se ha logrado cambiar sin problemas entre versiones de Android [51].

ESTADO DEL ARTE

Como se dijo, desde que Android tomó su posición prevalente en el mercado, los criminales han intentado vulnerar su seguridad; consecuentemente, los investigadores han intentado crear y mejorar las técnicas de detección.

Felt et al. [8] analizaron los incentivos, mostraron algunos *root exploits* en múltiples sistemas operativos (Android, Symbian, iOS) y estudiaron el comportamiento de las *payload* móviles en el 2011. Por su parte, Zhou y Jiang, quienes publicaron en 2012 una revisión [10] de *malware* en la plataforma Android y recolectaron un *dataset* de aplicaciones maliciosas de más de 1.200 muestras entre agosto de 2010 y octubre del 2011, consideran que es necesario desarrollar nuevas estrategias de detección, porque el software antivirus se desempeña pobremente en tareas de detección y es muy vulnerable a estrategias como reempaquetamiento –descargar aplicaciones no maliciosas populares, desensamblarlas, insertar cargas maliciosas en ellas y volverlas a subir a la tienda de aplicaciones oficial o a otras paralelas–.

En 2017, Tam et al. [9] enfocaron su trabajo en las técnicas disponibles para detección, como el análisis estático y dinámico, pero a diferencia de las anteriores revisiones, mencionan al aprendizaje de máquina como una herramienta efectiva para detección de *malware*. Esta revisión propone como direcciones de investigación futuras: el análisis híbrido y multinivel, la cobertura de código y los dispositivos híbridos con virtualización.

A través de la literatura, a partir de 2010 se encuentran múltiples tipos de algoritmos de *machine learning* para detección de *malware* en Android. En la Tabla 1 se presenta el desglose de cuáles algoritmos son utilizados de un total de 17 sistemas revisados (la suma es mayor que 17, porque algunos sistemas usan más de un algoritmo), en un rango que va de 2010 a 2017 e incluye algoritmos de aprendizaje supervisado y no supervisado.

Tabla 1. Algoritmos utilizados para la detección de malware en Android

Algoritmo	Sistemas usándolo (#)	Año
Deep learning	2	2014
SVM	6	2012, 2013, 2014
Random Forest	4	2012, 2013, 2017
Naive Bayes	5	2010, 2012, 2013, 2017
Perceptrón multicapa	1	2013
Regresión logística	5	2010, 2012, 2013, 2017
J48	5	2012, 2013, 2017
K-means	4	2011, 2012, 2013
K Nearest Neighbors (KNN)	2	2012
Otros modelos bayesianos	2	2013

Conforme pasa el tiempo, empiezan a aparecer los repositorios de *malware* y los investigadores se benefician de ellos. Si bien se han utilizado herramientas tanto de análisis estático como dinámico para obtener características, se puede notar que el estático es más usado. La precisión obtenida por los sistemas basados en aprendizaje de máquina considerados es siempre mayor a 85 %, lo que comparado con el obtenido por el método de firmas muestra tasas de detección superiores. A excepción de [52], los *datasets* son conformados por mínimo 200 ítems, Arp et al. [11], por ejemplo, consideran un *dataset* de 123.453 aplicaciones benévolas y 5.560 ítems de *malware*. En la Tabla 2 se resumen los sistemas encontrados. Como se evidencia en ella, se han probado varios tipos de algoritmos y analizado varias características en las aplicaciones.

Como se mencionó, una arquitectura debe ser compleja para aprender los patrones subyacentes en el *dataset* y evitar el *underfitting*, pero no tanto como para aprender de memoria el *dataset* de entrenamiento y terminar siendo un pobre predictor. Los parámetros de complejidad varían de algoritmo a algoritmo y

Tabla 2. Comparativo entre propuestas del estado del arte

Investigadores	Algoritmo	Decisiones arquitectónicas justificadas	Observaciones
Yuan et al. [53]	Deep learning	Sí, se prueban varias arquitecturas.	Extraen 200 características a partir de análisis estático y dinámico. Obtienen un 96 % de exactitud con datasets de malware real.
Arp et al. [11]	SVM lineal	No.	Realizan el análisis estático de un dataset de 123.453 apps benévolas y 5560 maliciosas. Tasa de detección: 95 %.
Amos, Turner y White [54]	Random Forest / Naive Bayes / Perceptrón multicapa / Red bayesiana / regresión logística / J48	No, una arquitectura para cada modelo.	Comparan los resultados obtenidos por varios algoritmos de ML y proponen un framework para validación de clasificadores para malware móvil.
Peiravian y Zhu [55]	SVM / J48	No se especifica.	Realizan análisis estático con las llamadas a la API y los permisos.
Mahindru y Singh [56]	Naive Bayes / J48 / Random Forest / regresión logística	No, se hacen varios experimentos con cada modelo.	Trabajan con un dataset con once mil aplicaciones; extraen 123 permisos dinámicos. Tasa de detección 95 %.
Sahs y Khan [57]	SVM de una clase	No se especifica.	Trabajan con un dataset con 2.081 aplicaciones benignas y 91 aplicaciones de malware. Tasa de detección: 95 %.
Rasthofer, Arzt, y Bodden [58]	SVM	Sí.	Se enfocan en clasificar fuentes y sifones de Android. Tasa de precisión: 92 %.
Aung y Zaw [59]	K-means / J48 / Random Forest	No se especifica.	Análisis estático de 200 aplicaciones. Tasa de precisión: 92 %.
Dini et al [60]	KNN	No se especifica.	Detector en tiempo real de malware que usa tanto análisis de kernel como a nivel de usuario. Tasa de detección cercana a 100 %.
Yerima et al. [61]	Modelo bayesiano	No se especifica.	De un dataset con mil aplicaciones benévolas y mil malignas, extraen 58 características. Tasa de detección cercana a 90 %.
Wu et al. [12]	K-means / KNN	Sí.	Realizan análisis estático usando llamados a API y permisos.

Tabla 2. Comparativo entre propuestas del estado del arte (cont.)

Investigadores	Algoritmo	Decisiones arquitectónicas justificadas	Observaciones
Sanz et al. [62]	Regresión logística / Naive Bayes / J48 / Random Forest	No se especifica.	Realizan análisis estático extrayendo permisos de un dataset con 1.811 aplicaciones benévolas y 249 maliciosas. Tasa de exactitud superior a 86 %
Wu y Hung [63]	SVM	No se especifica.	Realizan análisis estático y dinámico con un dataset formado por 32.000 aplicaciones benévolas y 32.000 malware. Tasa de exactitud mínima: 92.5 %.
Yuan, Lu, y Xue [64]	Deep learning	Sí.	Realizan análisis estático y dinámico con un dataset con 1.760 aplicaciones maliciosas y 20.000 benévolas. Tasas de detección: 96.76 %
Burguera, Zurutuza, y Nadjm-Tehrani [65]	K-means	Sí.	Trabajan con muestras de malware auto-escritas y reales. Tasa de detección cercana a 100 %
Gascon et al. [66]	SVM	No.	Usan grafos de llamadas. Tasa de detección: 89 %.
Shabtai [52]	Naive Bates	No se especifica.	Trabajan con un dataset compuesto por malware creado, pues indican que no había malware en Android detectado. Tasa de exactitud cercana a 90 %

su definición es crucial para los resultados obtenidos. Sin embargo, como se observa en la Tabla 2, no muchos sistemas se preocupan por probar múltiples hiperparámetros – o al menos por justificar su selección–. Es necesario desarrollar estudios que se refieran a la arquitectura adecuada que un cierto algoritmo debería tener para obtener resultados óptimos. También, sería muy valioso que el sistema de *machine learning* desarrollado sea abierto y pueda ser usado como componente para aplicaciones más grandes.

Para una actualización sobre investigaciones desarrolladas acerca de seguridad en Android, con énfasis en los análisis estático y dinámico y el aprendizaje de máquina, se recomienda revisar [4, pp. 36-40].

EL PROYECTO

METODOLOGÍA

Existen varias metodologías para realizar proyectos en el campo de la minería de datos, de ellas, para el desarrollo de este proyecto se consideraron: CRISP-DM (*Cross-Industry Standard Process for Data Mining*), SEMMA (*Sample, Explore, Modify, Model, Assess*) y ASUM (*Analytics Solution Unified Method*). Se decidió utilizar esta última porque al ser este un proyecto relacionado con una investigación de tipo académico, que no está orientada a solucionar un problema empresarial específico, está menos ligado a procesos organizacionales y no tan involucrado en un modelo de negocio.

ASUM se define como una guía paso a paso para llevar a cabo una implementación completa del ciclo de vida para soluciones analíticas IBM. Fue lanzado como una versión refinada y extendida de CRISP-DM en 2015 [67]. Tiene como ventajas: su mínimo riesgo, gracias a la experiencia de una compañía como IBM; la validación formal con pruebas estándar de la industria; su escalabilidad; y el hecho de que proporciona mapas de ruta para la implementación específicos para el producto [68].

Las fases de esta metodología son: analizar, esto es definir qué necesita lograr la solución (en términos de características y requisitos no funcionales); diseñar, es decir definir todos los componentes de la solución y sus dependencias, identificar recursos e instalar un ambiente de desarrollo; configurar, construir e integrar componentes basado en un enfoque iterativo e incremental; desplegar, o sea crear un plan para correr y mantener la solución; y operar y optimizar, con tareas de mantenimiento y puntos de chequeo que facilitan un empleo exitoso de la solución.

Para la construcción del software se decidió utilizar *Rational Unified Process* (RUP), metodología diseñada por la Rationale Software [85], por dos razones: la primera, que los requerimientos del proyecto están muy claramente definidos desde su inicio; la segunda, que a diferencia de un producto hecho a la medida, no habrán cambios mayores en la arquitectura o peticiones de módulos adicionales. Las principales características de la metodología RUP son: el desarrollo iterativo, la gestión de requerimientos, el empleo de arquitecturas basadas en componentes, el modelado visual del software, la verificación continua de la calidad y el control de cambios.

CREACIÓN DEL DATASET

La primera tarea para desarrollar el proyecto es crear el *dataset* para suministrarlo al algoritmo de *machine learning*, compuesto por aplicaciones maliciosas y no maliciosas. Durante una revisión de la literatura, se pudieron encontrar dos *datasets* de *malware* con artículos de investigación que los respaldan y certifican su legitimidad, estos son: el Android Malware Genome Project (AMGP) [69], que contiene más de 1.200 muestras de *malware* recolectadas por Zhou y Jiang [10]; y el *dataset* Drebin, proveído por la Technische Universitat Braunschweig [70], con 5.560 aplicaciones de 179 diferentes familias de *malware*. Las muestras de *malware* se obtuvieron de ellos. En cuanto a las aplicaciones benévolas, sus ejemplos fueron extraídos de las bases de datos Androsec, estudiadas por Urcuqui y Navarro [71].

Una vez los ejemplos del *dataset* son seleccionados, el siguiente paso es decidir cuáles características de cada ejemplo serán consideradas y cómo serán obtenidas. Debido a que algunos de los *datasets* especificados proveen las aplicaciones en el formato APK, es necesario usar una herramienta para extraer las características mencionadas. Se utilizó un descompilador, el APKTool [72] (Figura 4). Para su selección, el principal criterio fue que contara con una versión con línea de comandos, para poder usarlo múltiples veces a través de un *script* y procesar una gran cantidad de archivos automáticamente a través de comandos. Además, fue notorio que la mayoría de las opciones indicaba que usaban internamente el descompilador APKTool [72]. Por todo ello –y por la simplicidad del comando obtenido–, fue el elegido.

```
→ projects apktool d 0c859ad62b9dbccf8943fe4697f2a6b0cb917548.apk -o apk_test
I: Using Apktool 2.3.4 on 0c859ad62b9dbccf8943fe4697f2a6b0cb917548.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
S: WARNING: Could not write to (/home/danielajacome_/.local/share/apktool/framework), using /tmp instead...
S: Please be aware this is a volatile directory and frameworks could go missing, please utilize --frame-path if the default storage directory is unavailable
I: Loading resource table from file: /tmp/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Figura 4. Muestra de descompilación de la herramienta APKTool

En cuanto a la selección y extracción de características, como se ha expresado, la estrategia de seguridad más importante en Android es la API de permisos,

ella limita qué características del sistema pueden ser usadas por la aplicación y, lo que es más importante, deben ser aceptadas o denegadas por el usuario. Por ello fueron escogidas como las características que analizará el sistema desarrollado

ANÁLISIS PRELIMINAR

Con el fin de obtener alguna información previo al tratamiento de *machine learning* de la información, se realizaron algunos análisis preliminares. Para este propósito, se seleccionaron aleatoriamente 250 muestras de *malware* y 250 muestras de aplicaciones benignas, sus permisos fueron obtenidos desde el APK mediante descompilación. A partir de ahí, se calculó el promedio del número de permisos solicitados por cada tipo de aplicación y su desviación estándar (Tabla 3), con el fin de obtener algunas estadísticas acerca del problema.

Tabla 3. Permisos para datasets de malware y no malware

Tipo de aplicación	Promedio	σ
Malware	12.2	5.1
No malware	5	4.5
Total	8.6	6

El procedimiento realizado evidencia que las aplicaciones de *malware* solicitan más permisos que las no maliciosas. De hecho, hay una diferencia significativa entre los dos grupos, mediante una prueba t para varianzas diferentes, permitió obtener los siguientes resultados: $t(490) = 16.8, p < 0,001$. También hay una diferencia significativa entre los permisos que solicita cada tipo de aplicación, como se muestra en la Tabla 4, que corresponde a los diez permisos más solicitados.

Como se puede apreciar, hay diferencias muy marcadas en la cantidad de aplicaciones que requiere cada tipo de permiso, por ejemplo: el permiso *Read SMS*, que puede ser usado para robo de información, es solicitado por 24 aplicaciones benévolas, mientras que 156 maliciosas lo requieren (seis y media veces más); el permiso *Write SMS*, que puede ser usado para suscribir al usuario a servicios pagos, es solicitado nueve veces más por aplicaciones maliciosas que por aplicaciones benignas; y el permiso *Receive boot completed*, que se usa para

Tabla 4. Permisos más solicitados para aplicaciones Android

Permiso	Apps Benignas	Malware	Total
Internet	180	247	427
Access network state	109	209	318
Write external storage	120	169	289
Read phone state	180	247	427
Access WiFi state	47	164	211
Receive boot completed	58	135	193
Read SMS	24	156	180
Wake lock	75	92	167
Vibrate	62	99	161
Write SMS	15	136	151

detectar cuándo se enciende el celular –lo que facilita que la aplicación puede lanzar servicios que correrán en *background* y pasar inadvertidos–, es requerido dos veces más por aplicaciones maliciosas. Por otra parte, el permiso más requerido por ambos tipos de aplicación es Internet, necesario para navegar a través de las redes disponibles en un dispositivo. Las diferencias marcadas entre las dos clases permiten suponer que un sistema basado en *machine learning* puede desempeñarse bien en este tipo de clasificación al identificar estos patrones.

HERRAMIENTAS DE PROGRAMACIÓN

El segundo paso de la metodología seleccionada es el diseño, se trata de definir los componentes de la solución, sus dependencias y el ambiente de desarrollo. Actualmente, existen muchos lenguajes de programación que ofrecen herramientas para el manejo de complejos cálculos matemáticos y tareas relacionadas con el *machine learning*, los más populares son: Matlab, R, Java y Python.

Matlab es un lenguaje de programación propietario desarrollado por MathWorks, se basa en la manipulación de matrices y permite crear gráficos, usar algoritmos de optimización numérica y filtros para procesamiento digital de señales, entre otras funcionalidades. Existe también un programa libre similar a Matlab, llamado Octave.

R es un lenguaje para computación estadística y graficación, incluye una poderosa colección de herramientas gráficas para análisis de datos; para tareas

computacionalmente intensivas, puede ejecutar código C, C++ y Fortran en tiempo de ejecución; permite programación orientada a objetos usando las clases S3 y S4.

Java es un lenguaje de programación multipropósito basado en el paradigma orientado a objetos, posee WORA (*Write Once, Run Anywhere*), característica que permite que código escrito en Windows, Mac, Ubuntu o cualquier sistema operativo correrá en otro sistema que posea la máquina virtual Java. También ofrece herramientas de *machine learning* como Weka [73], creada por la Universidad de Waikato (Nueva Zelanda) que tiene una interfaz gráfica ideal para principiantes o puede importarse como librería.

Python es un lenguaje de programación multiparadigma de alto nivel con propósito general: cuenta con tipado dinámico y gestión automática de la memoria; soporta múltiples paradigmas de programación –orientado a objetos, imperativo, funcional, etc.–; y está disponible en casi cualquier sistema operativo. Sus extensas librerías de soporte constituyen una de sus principales ventajas, ellas permiten utilizarlo para una amplia gama de aplicaciones, como: programación web –el *framework* Django–, sistemas embebidos –las librerías mraa, paho-mqtt–, graficar funciones –matplotlib–, interfaces gráficas para escritor –tkinter–, y operaciones numéricas –numpy, scipy, pandas–. Las preguntas relacionadas con Python son las de mayor crecimiento, en comparación con las que se hacen sobre los otros grandes lenguajes de programación [74]. Python tiene además muchas librerías de *machine learning*, por ejemplo: Tensorflow, desarrollada por Google, que permite integración de la GPU [75]; PyTorch, que usa la librería Torch programada en C [76]; y Keras, una librería de alto nivel para trabajar con redes neuronales que corre sobre Tensorflow.

Por lo anterior –y por ser muy usado para *scripting*–, se seleccionó Python para desarrollar el sistema. Con el fin de tener más control sobre el algoritmo de propagación hacia adelante, la función de costo, el gradiente y su optimización, se decidió implementar el sistema con las librerías Pandas, Numpy y Scipy. Pandas es una librería de código abierto, con licencia BSD, que provee estructuras de datos fáciles de usar, con alto desempeño y funcionalidades adicionales como manejo de archivos y *parseo* de datos a partir de archivos; Numpy provee las herramientas básicas para computación científica (*e.g.*, manejo eficiente de operaciones de matrices, funciones de álgebra lineal, transformadas de Fourier, funciones de números aleatorios); y Scipy contiene rutinas amigables al usuario y eficientes para integración numérica y optimización.

PERCEPTRÓN MULTICAPA

Como se ha mencionado, la parte más difícil de la optimización de una red neuronal es ajustar los hiperparámetros de la red, principalmente porque no hay un algoritmo que indique de manera determinística: cuál es el número óptimo de neuronas en cada capa y cuál el número óptimo de capas o el parámetro de regularización adecuado; la mayoría de estos ajustes deben realizarse mediante ensayo y error. Además, la complejidad para cada problema es diferente dependiendo del contexto, del número de características consideradas y otros factores. Debido a esto, el primer paso de la investigación se realizó en una red neuronal de una sola capa, con el propósito de hallar cuáles son los hiperparámetros que otorgan los mejores resultados. Para manejar los cálculos, se construyó un sistema de RNA usando Python y las librerías Numpy y Scikit.

En primer lugar, se creó un *dataset* al unir mil muestras de *malware* e igual número de muestras de aplicaciones no *malware* escogidas aleatoriamente de los *dataset* mencionados. Posteriormente, los permisos fueron extraídos con ayuda del descompilador y un *script* que *parsea* los archivos AndroidManifest y extrae los permisos. Luego, esta información se consigna en un archivo csv de la siguiente manera:

- cada columna del archivo csv corresponde a un permiso (*e.g.*, Internet, Write SMS), a excepción de la última columna;
- cada fila del csv es una aplicación de ejemplo;
- si la aplicación i solicita el permiso j, el valor en la celda ij es 1, en caso contrario, su valor es 0;
- la columna final corresponde a la etiqueta real del ejemplo, que indica si la aplicación es *malware* o no.

Debido a que el *script* detectó que 128 permisos únicos fueron solicitados, las entradas son vectores de 128 dimensiones, compuestos por unos y ceros. De aquí, se determina el número de neuronas en las capas de entrada y salida: 128 neuronas en la capa de entrada y una en la de salida, debido a que es un clasificador binario. El sistema es entrenado y probado con varios valores del parámetro de regularización L2 (λ) y varios números de neuronas en la capa oculta, con el fin de encontrar los que otorguen mejores resultados. Los rangos considerados son: desde 0.2 hasta 2 en saltos de 0.2, es decir: 0.2, 0.4, ..., 2, para el parámetro de regularización; y desde 35 hasta 95, en saltos de 15, es decir: 35, 50, 65, 80 y 95, para el número de neuronas. En la Figura

5 se puede observar gráficamente el procedimiento realizado para probar el sistema y hallar los parámetros de mejor ajuste. Su descripción, paso a paso, se hace a continuación.

1. Se separa un 20 % de las etiquetas de ejemplo, respetando el porcentaje de distribución del *dataset* total. Este subconjunto es el *dataset* de prueba y se compone de 400 ejemplos.
2. Se divide el resto del *dataset* usando cinco *seeds* diferentes en 1200 ejemplos –el *dataset* de entrenamiento–, y 400 ejemplos –el *dataset* de validación–. Usando el primer número de neuronas y el primer parámetro de regularización (35 y 0.2, respectivamente) se obtienen los parámetros de mejor ajuste (Θ) a partir del conjunto de entrenamiento hallado. Con ellos se predicen las etiquetas del *dataset* de validación y se encuentra el F-score para cada partición (validación cruzada [18] [77]).
3. El procedimiento del paso anterior se repite para cada combinación de número de neuronas y λ. Las *seeds* son las mismas para cada iteración con el fin de estudiar las mismas cinco particiones con cada combinación.

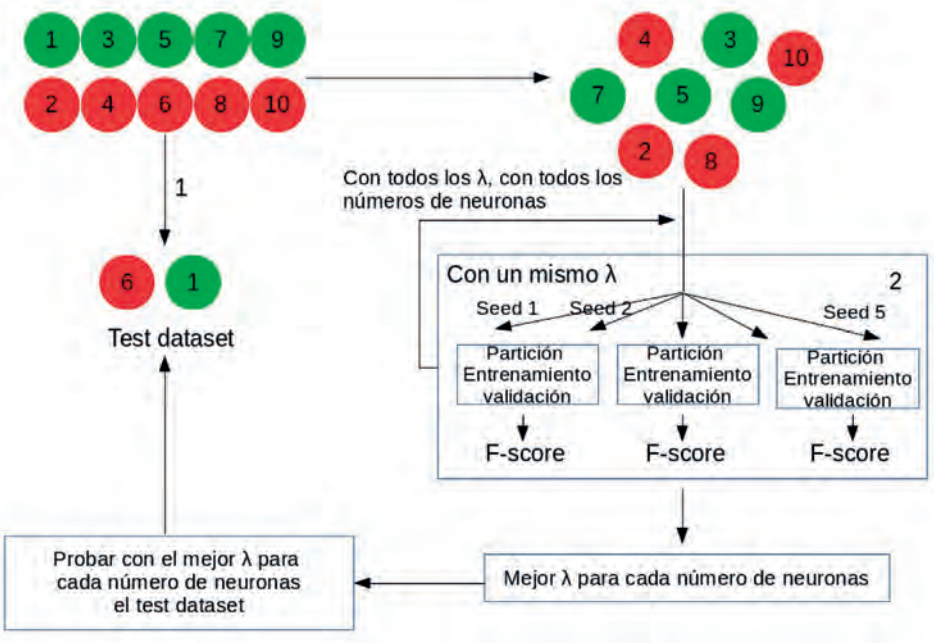


Figura 5. Procedimiento ejecutado para probar el sistema desarrollado

4. Se calcula un promedio de las cinco iteraciones por cada combinación de número de neuronas y parámetro de regularización y así se obtiene el parámetro de regularización que mejor se desempeña con cada número de neuronas. Este proceso se realizó varias veces porque diferentes particiones del *dataset* o la aleatorización del valor inicial de los pesos realizada al inicio de las pruebas pueden dar resultados diferentes.
5. Finalmente, la red neuronal se *testea* contra el *dataset* de prueba por cada número de neuronas, con el parámetro de regularización que arrojó mejores resultados. Así, también se va a encontrar el número de neuronas del rango mencionado, que mejor se desempeña.

Con el ambiente de desarrollo instalado, las arquitecturas a probar definidas y el proceso para hacerlo, se considera finalizada la etapa de diseño de la metodología. El puntaje F promedio resultante del proceso de validación cruzada realizado por cada combinación de número de neuronas y parámetro de regularización se muestra en la Tabla 5 (la sombra corresponde al mejor puntaje F obtenido).

Tabla 5. Resultados del puntaje F promedio en el dataset de validación con arquitecturas de RNA

λ	Neuronas en la capa escondida				
	35	50	65	80	95
0.2	97.49 %	97.2 %	97.39 %	97.29 %	97.44 %
0.4	97.49 %	97.55 %	97.6 %	97.54 %	97.49 %
0.6	97.31 %	97.21 %	97.06 %	97.11 %	94.63 %
0.8	97.21 %	96.8 %	96.76 %	96.7 %	97.02 %
1.0	96.61 %	96.66 %	96.71 %	96.09 %	96.53 %
1.2	96.31 %	96.25 %	96.31 %	96.29 %	96.14 %
1.4	96.24 %	96.15 %	96.04 %	96.03 %	96.03 %
1.6	95.99 %	95.99 %	95.77 %	95.94 %	95.94 %
1.8	96.14 %	95.73 %	95.87 %	95.73 %	95.62 %
2.0	95.42 %	95.73 %	95.39 %	95.76 %	95.08 %

En la Tabla 5, la sombra corresponde al mejor puntaje F obtenido. Se puede observar que el término de regularización que da consistentemente mejores resultados es 0.4. Debido a esto, λ se establece con ese valor para todas las

iteraciones de prueba. También, el máximo puntaje F se incrementa en el rango de 35 a 65 neuronas y, entonces, disminuye de nuevo, lo que evidencia que los sistemas de aprendizaje de máquina deben encontrar un nivel adecuado de complejidad de la hipótesis. Se espera entonces, que con 65 neuronas se obtengan los mejores resultados en las iteraciones de prueba.

Los resultados para las simulaciones de prueba se presentan en la Tabla 6. El mejor puntaje F en el *dataset* de prueba se obtiene, como se esperaba, con 65 neuronas, lo que confirma que la complejidad adecuada de la red neuronal está alrededor de este número. Los resultados de precisión y sensibilidad indican que el sistema presenta más falsos negativos (instancias etiquetadas como benévolas que realmente sí lo son) que falsos positivos (instancias etiquetadas como *malware* que realmente no lo son). Es un tema abierto si es más importante lograr una tasa muy baja de falsos positivos o de falsos negativos. En cualquier caso, al variar el umbral usado por las neuronas para decidir si la entrada es 0 o 1 –en estas simulaciones se escogió 0.5–, se puede alterar el intercambio entre precisión y sensibilidad.

Tabla 6. Resultados del puntaje F promedio en el dataset de prueba con arquitecturas de RNA de una capa oculta

Neuronas en la capa escondida	Escenario	Precisión (%)	Sensibilidad (%)	Puntaje F (%)
35	Entrenamiento	99.14	97.96	98.55
	Prueba	98.47	95.54	96.98
50	Entrenamiento	99.14	97.79	98.46
	Prueba	98.47	95.54	96.98
65	Entrenamiento	99.48	97.79	98.63
	Prueba	98.97	95.54	97.23
80	Entrenamiento	99.14	97.96	98.55
	Prueba	98.46	95.05	96.73
95	Entrenamiento	99.14	97.96	98.55
	Prueba	98.46	95.05	96.73

De acuerdo con la definición de Sze et al. [34], se puede decir que las de dos capas son redes neuronales profundas (*deep neural networks*). Añadir capas agrega no linealidad a las redes neuronales de una sola capa, debido a que se realiza un producto de las entradas consigo mismas cada vez que los datos pasan de

una capa a otra. En consecuencia, se espera que pueda encontrar patrones más complejos en los datos que los hallados por una red de una sola capa; sin embargo, al igual que con el anterior tipo de redes, no es posible saber qué hiperparámetros darán los mejores resultados, por lo que se probaron diferentes números de capas con múltiples números de neuronas, buscando hallar cuáles parámetros –de arquitectura y de regularización–, dan los mejores resultados.

El proceso es el mismo utilizado para el perceptrón multicapa de una sola capa oculta. Los números de capas y arquitecturas consideradas para dos capas ocultas son: 65 y 35 neuronas, 35 y 65 neuronas, 70 y 10 neuronas, 10 y 70 neuronas; y para tres capas ocultas: 95, 65 y 35 neuronas; 80, 50 y 35 neuronas; 65, 65 y 65 neuronas.

Los resultados de las iteraciones con los conjuntos de validación, en el caso de dos capas ocultas, se muestran en la Tabla 7. Una conclusión interesante que puede ser obtenida de esta tabla es que el parámetro de regularización óptimo es, en todos los casos considerados, mayor al obtenido en el caso de una sola capa oculta, lo que se puede ser porque la complejidad de la hipótesis es superior que en el ejemplo anterior, y por lo tanto, el parámetro de regularización debería ser mayor para evitar el *overfitting*.

Del mismo modo, al comparar las Tablas 5 y 7 es posible afirmar que los puntajes F de validación obtenidos son, en general, mayores que los obtenidos

Tabla 7. Resultados del puntaje F en el dataset de validación con arquitecturas de RNA de dos capas ocultas

λ	Combinación de neuronas en la primera y segunda capas ocultas			
	35, 65	65, 35	70, 10	10, 70
0.2	98.25 %	98.36 %	96.04 %	97.67 %
0.4	98.23 %	98.6 %	95.81 %	97.9 %
0.6	97.72 %	97.92 %	96.08 %	97.89 %
0.8	98.48 %	96.98 %	96.08 %	97.66 %
1.0	98.48 %	97.68 %	95.44 %	97.44 %
1.2	98.23 %	97.68 %	95.58 %	97.66 %
1.4	97,00 %	96.73 %	95.85 %	97.43 %
1.6	97.49 %	96.49 %	95.58 %	97.21 %
1.8	97.98 %	96.34 %	95.31 %	96.73 %
2.0	97.73 %	96.24 %	94.79 %	96.73 %

con una red neuronal de una sola capa. El mejor puntaje F para los *datasets* de validación fue obtenido con la arquitectura 65-35. También debe notarse que en las arquitecturas 70-10 y 10-70 se obtiene un puntaje F inferior que con las arquitecturas 35-65 y 65-35. Cada red fue entrenada con el mejor λ encontrado en esta sección.

Los resultados respecto al *dataset* de prueba se encuentra en la Tabla 8. Esta tabla indica que la arquitectura más apropiada para redes neuronales artificiales de dos capas es 65 y 35 neuronas en cada capa. Aún así, la arquitectura 35-65 produce resultados comparables, de tal manera que los hiperparámetros óptimos pueden estar alrededor de estos números.

Tabla 8. Resultados del puntaje F promedio en el dataset de prueba con arquitecturas de RNA de dos capas ocultas

Arquitectura de la red	Escenario	Precisión (%)	Sensibilidad (%)	Puntaje F (%)
35, 65	Entrenamiento	99.14	97.96	98.55
	Prueba	98.98	95.59	97.26
65, 35	Entrenamiento	100	98.42	99.2
	Prueba	100	97.44	98.7
10, 70	Entrenamiento	99.82	94.95	96.5
	Prueba	98.1	94.95	96.5
70, 10	Entrenamiento	98.1	98.63	99.22
	Prueba	97.57	95.71	96.63

Por último, se repite el proceso con tres capas ocultas, con el fin de hallar el mejor parámetro de regularización y la arquitectura óptima en este escenario. Al comparar las Tablas 7 y 9, es claro que los valores del puntaje F son inferiores para tres que para dos capas; esto puede suceder por el incremento en la complejidad que implica la capa extra. El valor sombreado corresponde al mejor parámetro de regularización para cada configuración. Estos son los valores con los que se debe probar el sistema contra el *dataset* de prueba, buscando la mejor arquitectura posible. A continuación, el sistema se entrenó con los pesos optimizados y se calcula la precisión, el *recall* y el puntaje F para el *dataset* de prueba, los resultados se presentan en la Tabla 10.

Al comparar puntaje F del *dataset* de prueba con tres y dos capas, es claro que disminuye con tres, lo que probablemente se debe a que es una hipótesis

Tabla 9. Resultados del puntaje F promedio en el dataset de validación con arquitecturas de RNA de tres capas ocultas

λ	Combinación de neuronas en la primera, segunda y tercera capas ocultas		
	35, 65	70, 10	10, 70
0.2	96.27 %	97.17 %	97.52 %
0.4	96.85 %	97.94 %	97.97 %
0.6	97.07 %	97.67 %	98.49 %
0.8	96.60 %	97.67 %	97.20 %
1.0	96.38 %	96.90 %	97.98 %
1.2	95.90 %	96.92 %	97.72 %
1.4	96.14 %	97.44 %	97.19 %
1.6	96.14 %	96.92 %	96.46 %
1.8	96.14 %	96.92 %	96.41 %
2.0	96.40 %	94.60 %	95.70 %

Tabla 10. Resultados del puntaje F promedio en el conjunto de prueba con arquitecturas de RNA con tres capas ocultas

Arquitectura de la red	Escenario	Precisión (%)	Sensibilidad (%)	Puntaje F (%)
95, 65, 35	Entrenamiento	99.64	98.44	99.04
	Prueba	98.99	95.17	97.04
80, 50, 35	Entrenamiento	99.83	98.85	99.34
	Prueba	97.43	96.45	96.94
65, 65, 65	Entrenamiento	100	98.66	99.33
	Prueba	97.51	96.55	97.03

muy compleja y a que la no linealidad introducida con esta nueva capa puede ser excesiva. En este caso, para incrementar el puntaje F obtenido se podría disminuir el número de neuronas por capa o aumentar el parámetro de regularización a más de 2.0. Los mejores resultados para este tipo de redes se obtuvieron con las opciones de 96, 65 y 35 neuronas y la de 65, 65 y 65 neuronas. Estas dos opciones tienen un promedio de 65 neuronas por capa, a diferencia de la opción de 80, 50 y 35 neuronas. Por tanto, puede confirmarse la intuición inicial de que el número de 65 neuronas por capa es el indicado para este tipo de aplicación.

EVALUACIÓN

Como estaba previsto en los objetivos, la evaluación se realizó comparando el desempeño del sistema frente a enfoques tradicionales, como métodos de firma, y con otros sistemas basados en aprendizaje de máquina descritos en el estado del arte. El procedimiento usual de comparación de resultados de diferentes sistemas de aprendizaje de máquina es realizar un *benchmark*, es decir, probar un producto o servicio contra un punto de referencia existente. Para esto, es deseable comparar los sistemas construidos en igualdad de condiciones, esto es, con el mismo *dataset*.

Para aplicaciones como reconocimiento de imágenes, existen *datasets* muy populares contra los que este tipo de aplicaciones se prueban regularmente, tales como el MNIST [78] y el CIFAR [79]. No obstante, para la detección de *malware* en Android no hay aún un *dataset* estándar contra el cual probar los sistemas desarrollados con propósitos de comparación, como se puede evidenciar en los *datasets* citados en el estado del arte.

En la Tabla 11 se presenta el *dataset* utilizado y el resultado obtenido por algunos sistemas del estado del arte. Los resultados "más comparables" con los obtenidos con el sistema desarrollado son los que utilizan los mismos *datasets*: AMGP y Drebin. Comparándolo con las estrategias de detección tradicional [10] que usan el AMGP, el sistema desarrollado presenta mejor desempeño: usando el mismo *dataset* de *malware*, alcanza un 98.7 % de exactitud.

También se puede notar que los algoritmos de *machine learning* mostrados en la Tabla 11 obtienen mejores resultados que los sistemas tradicionales en todos los casos, por lo tanto, esta estrategia de detección es muy promisoria, por lo que se alienta a realizar más investigación en este tema.

CONCLUSIONES Y TRABAJO FUTURO

Se realizó un análisis preliminar de los permisos para aplicaciones maliciosas y no maliciosas, así se evidenció que los permisos proveen información suficiente para clasificar un ejemplo como *malware*, debido a que hay una diferencia significativa entre el número y tipo de permisos solicitados por cada tipo de aplicación. Este análisis corresponde a la primera etapa de la metodología ASUM-DM.

Tabla 11. Comparativo de mejores resultados: sistema desarrollado / herramientas del estado del arte

Sistema	Exactitud (%)	Método implementado	Dataset usado
Línea base	50,00	-	-
Mejor caso de antivirus móvil existente [10]	79.60	-	AMGP [69]
Yuan et al. [53]	96.50	*Deep learning*	Contagio mobile mini dump [80]
Arp et al. [11]	94,00	SVM	Drebin [87]
Amos, Turner y White [54]	94.53	Múltiples algoritmos de ML	AMGP
Mahindru y Singh [56]	99.60	Múltiples algoritmos de ML	Varios, entre ellos AMGP
Aung y Zaw [59]	91.75	Múltiples algoritmos de ML	Se descargan apps maliciosas y no maliciosas de la tienda de apps.
Yerima et al. [61]	92.10	Modelo bayesiano	AMGP
Sanz et al. [62]	86.41	Múltiples algoritmos de ML	Se descargan las aplicaciones de la tienda a través de una API
Yuan, Lu, y Xue [64]	96.76	*Deep learning*	AMGP y Contagio
Sistema desarrollado	98.70	RNA y *Deep learning*	AMGP y Drebin

La segunda fase de la metodología se aplicó al seleccionar las herramientas para construir el sistema, junto con las arquitecturas a probar. A su vez, como resultado de la tercera fase de la metodología se construyó un sistema de aprendizaje de máquina escrito en Python, este sistema no está diseñado exclusivamente para esta aplicación particular (seguridad en Android), sino que puede ser usado en cualquier *dataset* que se ajuste al formato csv especificado. De hecho, el sistema fue creado como un *script* de manera que el usuario pueda especificar el número de capas y el número de neuronas por capa y la simulación se realice en segundo plano. Dado que el sistema es académico, no se definió un entorno de producción, así que la fase de despliegue de la metodología consta de realizar las simulaciones en el entorno local de desarrollo. Por último, el resultado de la fase operar y optimizar es un estudio del ajuste y los parámetros óptimos para esta aplicación en particular. Se realizaron varias simulaciones, con diferentes arquitecturas de red y parámetros de regularización, con el fin de encontrar la región donde se encuentran los parámetros de mejor ajuste. Estos análisis fueron realizados para un perceptrón multicapa sencillo y para redes

neuronales profundas con dos y tres capas ocultas. A medida que se agregaron capas ocultas, el procesamiento fue más intensivo y tomó más tiempo: para la red neuronal de una sola capa oculta, tardó de uno a dos minutos entrenar a la red; con dos capas, el tiempo de entrenamiento aumentó a un rango de 5 a 8 minutos; y con tres capas, el entrenamiento gastó aproximadamente diez minutos. Hay algunas técnicas que permiten mayor velocidad, como son la aceleración con GPU y el cálculo del gradiente con *mini-batches*.

La mejor arquitectura para esta aplicación particular estuvo consistentemente alrededor de las 65 neuronas por capa: con una capa oculta, el número de mejor desempeño fue 65; con dos capas ocultas, fue 65 en la primera capa y 35 en la segunda; y con tres capas ocultas fueron 95, 65 y 35 neuronas y 65, 65 y 65 neuronas. Este estudio abarcó también el parámetro de regularización L2 (λ). En el caso de una capa oculta, el mejor parámetro de regularización encontrado fue 0.4, para todos los números de neuronas en la capa oculta utilizados; al aumentar la complejidad, el parámetro de regularización de mejores resultados aumenta también, ya que es necesario penalizar más las hipótesis más complejas.

Debido a que el sistema fue creado para trabajar como un *script* y a que es altamente personalizable, puede ser usado como componente en un sistema más grande. Un resultado adicional, que no estaba previsto en los objetivos fue la creación de un sistema web utilizando tecnologías como Ruby on Rails y peticiones remotas para permitir acceso al sistema de aprendizaje de máquina construido a cualquier persona.

Aunque este sistema está aún en una fase preliminar, se espera que pueda recibir información de múltiples posibles campos de aplicación, no únicamente Android, y que permita la democratización del acceso a las herramientas de *machine learning*.

Como trabajo a futuro se propone:

- incluir más características de las aplicaciones para aumentar las tasas de detección, lo que puede ser realizado, por ejemplo, incluyendo análisis dinámico y tráfico web enviado y recibido por la aplicación;
- intentar más combinaciones de número de capas y número de neuronas por capa, con el fin de encontrar si alguna combinación aún no probada da mejores resultados que lo encontrado en este proyecto;
- intentar con diferentes tipos de sinapsis, en este trabajo, sólo se consideró

sinapsis completamente conectadas, sin embargo, una mejor clasificación podría lograrse usando sinapsis de estilo convolucional o recurrente;

- desplegar el sistema construido como un servicio en Internet, de manera que los celulares puedan enviar una petición con la información de cierta aplicación y el servicio conteste con la clasificación correspondiente (es o no es *malware*), de esta manera, la carga pesada del procesamiento no sería realizada en el celular, sino en un servidor que, por definición, tiene mayores capacidades para hacerlo; y
- invertir más recursos en el sistema web citado, de tal manera que la gente pueda utilizar aprendizaje de máquina en sus proyectos personales.

El sistema basado en aprendizaje de máquina desarrollado podría ser útil en la lucha contra el cibercrimen y para proteger a los ciudadanos de esta amenaza, especialmente en un país como Colombia, donde un gran porcentaje de la población no está muy enterada de los riesgos que la tecnología conlleva y podría ser presa fácil de cibercriminales (sin contar con que la regulación en este tema no está muy avanzada).

También se puede concluir que con las tecnologías actuales es muy factible realizar investigación en aprendizaje de máquina y que no se necesita un computador de alta gama para este propósito. Con un computador de media gama, estos cálculos pueden completarse en tiempos razonables (4 a 6 minutos). También, los computadores de hoy en día casi siempre tienen una unidad de procesamiento gráfico (GPU) que pueden acelerar el proceso de encontrar mínimos y optimizar la función de costo.

Una observación final es que el aprendizaje de máquina, después de un período de 'invierno', se está convirtiendo en una de las ramas de las ciencias de la computación más populares, y está siendo utilizada para una gran cantidad de problemas. Para esta aplicación en particular, y de acuerdo con los resultados de la revisión de la literatura y los del sistema desarrollado, hay una mejora notable en el desempeño cuando se usa un enfoque de *machine learning* para clasificar *malware* en Android.

REFERENCIAS

[1] J. Wannstrom. (Jun. 2013). LTE-Advanced. [Online]. Disponible: http://www.3gpp.org/technologies/keywords-acronyms/97-lte-advanced

[2] A. Spadafora. (Feb. 2017). ITU reveals 5G specs including 20 Gbps download speeds and 4 ms latency [Online]. Disponible: https://www.itproportal.com/news/itu-reveals-5g-specs- including-20gbps-download-speeds-and-4ms-latency/

[3] "Smartphone market share." (Jun. 22, 2020). Distribuido por IDC. https://www.idc.com/promo/smartphone-market-share/os

[4] C. Urcuqui, M. García, J. L. Osorio, y A. Navarro, *Ciberseguridad: un enfoque desde la ciencia de datos*. Cali, Colombia: Editorial Universidad Icesi, 2018.

[5] "The economic impact of cybercrime: no slowing down," MCAfee, Santa Clara, CA, 2017. https://www.mcafee.com/enterprise/en-us/assets/executive-summaries/es-economic-impact-cybercrime.pdf

[6] B. Toledano. (May. 12, 2017). El ciberataque con el virus wannacry se extiende a nivel mundial [Online]. Disponible: https://www.elplural.com/el-telescopio/tech/el-ciberataque-del-virus-wannacry-se-extiende-a-nivel-mundial_103293102

[7] (May. 13, 2017). Cyber attack: Europol says it was unprecedented in scale. [Online]. Disponible: https://www.bbc.com/news/world-europe-39907965#:~:text=A%20cyber%2Dattack%20that%20hit,%22to%20identify%20the%20culprits%22

[8] A. P. Felt, M. Finifter, E. Chin, S. Hanna, y D. Wagner, "A survey of mobile malware in the wild," en *Proceedings of the 1st ACM workshop on security and privacy in smartphones and mobile devices*, 2011, pp. 3–14.

[9] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, y L. Cavallaro, "The evolution of Android malware and Android analysis techniques," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, p. 76, 2017.

[10] Y. Zhou y X. Jiang, "Dissecting Android malware: Characterization and evolution," en *Security and Privacy (SP), 2012 IEEE Symposium on*, 2012, pp. 95–109.

[11] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, y K. Rieck, "Drebin: Effective and explainable detection of Android malware in your pocket," en *2014 NDSS Symposium, 2014*, ses. 11, pon. 3.

[12] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, y K.-P. Wu, "Droidmat: Android malware detection through manifest and API calls tracing," en *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*, 2012, pp. 62–69.

[13] P. Teufl, M. Ferk, A. Fitzek, D. Hein, S. Kraxberger, y C. Orthacker, "Malware detection by applying knowledge discovery processes to application metadata on the Android market (Google Play)," *Security and Communication Networks*, vol. 9, no. 5, pp. 389–419, 2016.

[14] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.

[15] L. A. Cabrera, M. E. Elbuluk, y I. Husain, "Tuning the stator resistance of induction motors using artificial neural network," *IEEE Transactions on Power Electronics*, vol. 12, no. 5, pp. 779–787, 1997.

[16] Y.-Y. Hsu y C.-R. Chen, "Tuning of power system stabilizers using an artificial neural network," *IEEE Transactions on Energy Conversion*, vol. 6, no. 4, pp. 612–619, 1991.

[17] M. H. Esfe, S. Saedodin, N. Sina, M. Afrand, y S. Rostami, "Designing an artificial neural network to predict thermal conductivity and dynamic viscosity of ferromagnetic nanofluid," *International Communications in Heat and Mass Transfer*, vol. 68, pp. 50–57, 2015.

[18] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," en IJCAI'95: Proceedings of the 14th International Joint Conference on Artificial Intelligence, ago. 1995, vol. 2, pp. 1137–1143

[19] A. M. Turing, "Computing machinery and intelligence," in *Parsing the Turing Test*, The Netherlands: Springer, 2009, pp. 23–65.

[20] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological Review*, vol. 65, no. 6, p. 386, 1958.

[21] Y. LeCun, L. Bottou, Y. Bengio, y P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[22] E. Rosten y T. Drummond, "Machine learning for high-speed corner detection," en *Computer Vision ECCV 2006* [Lecture Notes in Computer Science, vol. 3951], Berlín, Heidelberg: Alemania, 2006, pp. 430–443, doi:10.1007/11744023_34

[23] B. Pang, L. Lee, y S. Vaithyanathan, "Thumbs up?: sentiment classification using machine learning techniques," in *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing*, 2002, vol. 10, pp. 79–86.

[24] J. Bennett y S. Lanning, "The Netflix prize," en *Proceedings of KDD Cup and Workshop*, 2007, pp. 35–39.

[25] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[26] C. J. Watkins y P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[27] D. H. Hubel y T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of Physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[28] A. Krizhevsky, I. Sutskever, y G. E. Hinton, "Imagenet classification with deep convolutional neural networks," en *Advances in Neural Information Processing Systems Conference, 2012*, pp. 1097–1105.

[29] P. Y. Simard, D. Steinkraus, y J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," en Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings, 2003, pp. 958-963, doi: 10.1109/ICDAR.2003.1227801.

[30] A. v.d. Oord, S. Dieleman, y B. Schrauwen, "Deep content-based music recommendation," en *Advances in Neural Information Processing Systems Conference, 2013*, pp. 2643–2651.

[31] R. Collobert y J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," en *Proceedings of the 25th International Conference on Machine Learning*, 2008, pp. 160–167.

[32] H. Sak, A. Senior, y F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," en *15th Annual Conference of the International Speech Communication Association*, 2014, pp. 338-342.

[33] S. Fernández, A. Graves, and J. Schmidhuber, "An application of recurrent neural networks to discriminative keyword spotting," en *International Conference on Artificial Neural Networks ICANN, 2007* [LNCS, vol. 4669], Berlín-Heildelberg, Springer, 2007, pp. 220–229.

[34] V. Sze, Y. Chen, T.-J. Yang, y J. S. Emer, "Efficient processing of deep neural networks: a tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[35] "Neural network basics," TUAS, Tampere. Finlandia, Neurocomputing (TIETS07), Nov. 2019. [Online]. http://www.uta.fi/sis/tie/neuro/index/Neurocomputing2.pdf

[36] Google Developers. (Feb. 10, 2020). Regularización para lograr la simplicidad: regularización l2 [Online]. Disponible: https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/l2-regularization?hl=es

[37] P. Werbos, *Beyond regression: new tools for prediction and analysis in the behavioral sciences*, Cambridge, MA: Harvard University, 1975.

[38] Developers Android. (Ago. 17, 2020). Guide to app architecture [Online]. Disponible: https://developer.android.com/jetpack/guide

[39] Developers Android. (Jun. 15, 2020). Security tips [Online]. Disponible: https://developer.android.com/training/articles/security-tips

[40] Developers Android. (Sep. 1, 2020). Zona de pruebas de la aplicación [Online]. Disponible: https://source.android.com/security/app-sandbox

[41] A. S. Hornby, A. P. Cowie, y J. W. Lewis, *Oxford advanced learner's dictionary of current English*. Londres, UK: Oxford University Press, 1974.

[42] C. Schmidt. "The what, why and how of the 1988 Internet worm." Snowplow. https://snowplow.org/tom/worm/worm.html

[43] S. Arzt, S. Rasthofer, y E. Bodden, "The soot-based toolchain for analyzing Android apps," en *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*, 2017, pp. 13–24.

[44] "Decompile-apk." (Ene. 2020). Distribuido por Github. https://github.com/venshine/decompile-apk

[45] "Apk decompiler app," https://www.apkdecompilers.com/ (acceso: oct. 22, 2018).

[46] "Android apk decompiler," http://www.decompileandroid.com/ (acceso: oct. 22, 2018).

[47] T. Azim and I. Neamtiu, "Targeted and depth-first exploration for systematic testing of Android apps," *ACM Sigplan Notices*, vol. 48, no. 10, pp. 641–660, 2013.

[48] R. Unuchek. (2013). "The most sophisticated Android trojan" [Online]. Disponible: https://securelist.com/the-most-sophisticated-android-trojan/35929/

[49] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. Van Der Veen, y C. Platzer, "Andrubis–1,000,000 apps later: a view on current Android malware behaviors," in *Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS), 2014 Third International Workshop on*, 2014, pp. 3–17.

[50] L.-K. Yan y H. Yin, "Droidscope: seamlessly reconstructing the OS and dalvik semantic views for dynamic Android malware analysis." en *USENIX Security Symposium*, 2012, pp. 569–584.

[51] K. Tam, S. J. Khan, A. Fattori, y L. Cavallaro, "Copperdroid: automatic reconstruction of Android malware behaviors," en *NDSS Symposium, 2015*, ses. 2, pon. 4.

[52] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, y Y. Weiss, "Andromaly: a behavioral malware detection framework for Android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161–190, 2012.

[53] Z. Yuan, Y. Lu, Z. Wang, y Y. Xue, "Droid-sec: deep learning in Android malware detection," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 371–372, 2014.

[54] B. Amos, H. Turner, y J. White, "Applying machine learning classifiers to dynamic Android malware detection at scale," en *IWCMC 2013, Wireless Communications and Mobile Computing Conference*, 2013, pp. 1666–1671.

[55] N. Peiravian y X. Zhu, "Machine learning for Android malware detection using permission and API calls," en ICTAI 2013, 25th International Conference on Tools with Artificial Intelligence, 2013, pp. 300–305.

[56] A. Mahindru y P. Singh, "Dynamic permissions based Android malware detection using machine learning techniques," en *Proceedings of the 10th Innovations in Software Engineering Conference*, 2017, pp. 202–210.

[57] J. Sahs y L. Khan, "A machine learning approach to Android malware detection," en *European Intelligence and Security Informatics Conference, EISIC 2012*, pp. 141–147.

[58] S. Rasthofer, S. Arzt, y E. Bodden, "A machine-learning approach for classifying and categorizing Android sources and sinks," en *NDSS Symposium, 2014*, ses. 10, pon. 1.

[59] Z. Aung y W. Zaw, "Permission-based Android malware detection," *International Journal of Scientific & Technology Research*, vol. 2, no. 3, pp. 228–234, 2013.

[60] G. Dini, F. Martinelli, A. Saracino, y D. Sgandurra, "Madam: a multi-level anomaly detector for Android malware," en *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security* [Lecture Notes in Computer Science, vol. 7531], Berlín-Heidelberg, Alemania: Springer, 2012, pp. 240–253.

[61] S. Y. Yerima, S. Sezer, G. McWilliams, y I. Muttik, "A new Android malware detection approach using bayesian classification," en *AINA 2013, IEEE 27th International Conference on Advanced Information Networking and Applications*, 2013, pp. 121–128.

[62] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, y G. Álvarez, "Puma: permission usage to detect malware in Android," en *International Joint Conference CISIS'12-ICEUTE´12-SOCO´12: special sessions*, Berlín-Heidelberg, Alemania: Springer, 2013, pp. 289–298.

[63] W.-C. Wu y S.-H. Hung, "Droiddolphin: a dynamic Android malware detection framework using big data and machine learning," en *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*, 2014, pp. 247–252.

[64] Z. Yuan, Y. Lu, y Y. Xue, "Droiddetector: Android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114–123, 2016.

[65] I. Burguera, U. Zurutuza, y S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for Android," en *Proceedings of the 1st ACM workshop on security and privacy in smartphones and mobile devices*, 2011, pp. 15–26.

[66] H. Gascon, F. Yamaguchi, D. Arp, y K. Rieck, "Structural detection of Android malware using embedded call graphs," en *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*. 2013, pp. 45–54.

[67] T. Zeutschler. (2016). IT applications in business analytics [Power Point Slides]. Disponible: https://wiwi.hs-duesseldorf.de/personen/thomas.zeutschler/Documents/HSD_W_ITAiBA_Zeutschler_SS2016_Lecture3_ToolsTechnologies.pdf

[68] "Analytics solutions unified method: implementations with Agile principles," IBM, Armonk, NY, 2016. [Online]. Disponible: ftp://ftp.software.ibm.com/software/data/sw-library/services/ASUM.pdf

[69] J. Zhou y X. Jiang, "Android malware genome project." (2018). Distribuido por: Noth Caroline State University. http://www.malgenomeproject.org/

[70] "The Debrin dataset." (2016). Distribuido por: Technische Univeristat Braunschweig, https://www.sec.cs.tu-bs.de/~danarp/drebin/

[71] C. Urcuqui y A. Navarro, "Machine learning classifiers for Android malware analysis," en *Communications and Computing (COL- COM), 2016 IEEE Colombian Conference on*, 2016, doi:10.1109/ColComCon.2016.7516385.

[72] R. Wiśniewski´y C. Tumbleson, "A tool for reverse engineering." (Abr. 26, 2018). Distribuido por Github. https://ibotpeaches.github.io/Apktool/

[73] I. H. Witten, E. Frank, L. E. Trigg, M. A. Hall, G. Holmes, y S. J. Cunningham, "Weka: practical machine learning tools and techniques with Java implementations," Univ. of Wikato, Hamilton, New Zealand, Working Paper 99/11, 1999.

[74] "Top 3 most popular programming languages in 2018." Finaxe.net. http://finaxe.net/2018/08/31/top-3-most-popular-programming-languages-in-2018-and-their-annual-salaries/ (acceso: oct. 22, 2018).

[75] Tensorflow. https://www.tensorflow.org/?hl=es (acceso: oct. 22, 2018).

[76] Pytorch. https://pytorch.org/(acceso: oct. 22, 2018).

[77] R. R. Picard y R. D. Cook, "Cross-validation of regression models," *Journal of the American Statistical Association*, vol. 79, no. 387, pp. 575–583, 1984.

[78] C. C. LeCun, Yann y C. Burges, "The MNIST database of handwritten digits," http://yann.lecun.com/exdb/mnist/

[79] A. Krizhevsky y G. Hinton. (2009). Learning multiple layers of features from tiny images. [Online]. Disponible: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[80] "Contagio mobile malware mini dump," http://contagiominidump.blogspot.com/

Búsqueda local distribuida basada en la exploración de vecindarios en paralelo para el problema RDCMST

César Loaiza, MSc.
Gabriel Tamura Morimitsu, Ph.D

Citación

C. Loaiza y G. Tamura, "Búsqueda local distribuida basada en la exploración de vecindarios en paralelo para el problema RDCMST," en *Aprovisionamiento ágil – Clasificación de malware – Optimización Giraph* [Bitácoras de la maestría, vol. 7], Cali, Colombia: Universidad Icesi, 2020, pp. 167-250.

RESUMEN

RDCMST (*Rooted Distance-Constrained Minimum Spanning Tree*) es un problema de optimización complejo (NP-hard) cuya solución se puede aplicar, entre otros campos, al diseño de redes de telecomunicaciones. Investigaciones previas para abordarlo proponen soluciones que van desde algoritmos exactos, como los modelos clásicos de programación lineal, hasta métodos heurísticos que incluyen el uso de búsquedas locales, pero tiene al menos dos carencias importantes: no existen aproximaciones paralelas que hayan sido diseñadas para aprovechar varias unidades de procesamiento y las aproximaciones existentes se limitan a instancias de unos pocos miles de vértices. Este proyecto se enfocó en la construcción de una estrategia distribuida para resolver el problema a partir de una búsqueda local que realiza una exploración paralela del vecindario, la cual fue implementada en un software distribuido que permite lidiar con instancias del problema de cientos de miles de vértices. Para lograrlo, se afrontaron retos asociados principalmente con el desempeño de la solución y el uso adecuado de los recursos computacionales disponibles, la mayoría de ellos se superó con la adopción Apache Giraph, un *framework* para procesamiento de grafos, lo que a su vez permitió estudiar qué tan adecuado es su uso para resolver problemas similares. Además de la estrategia para resolver RDCMST y su implementación en un ambiente distribuido, este trabajo incluye una serie de algoritmos diseñados para su ejecución en paralelo en distintos nodos de procesamiento, que resuelven problemas comunes en el diseño de soluciones a problemas de grafos. Una evaluación experimental de la estrategia mostró su capacidad para manejar instancias de decenas de miles de vértices en un tiempo razonable y dio indicios de funcionar mejor con instancias más grandes y de tener un mejor desempeño bajo ciertas condiciones de la solución subóptima inicial en la búsqueda local.

INTRODUCCIÓN

Los problemas de optimización se pueden definir, de manera simple, como encontrar y elegir una solución que maximice un conjunto de criterios dentro de un conjunto de posibles soluciones. La búsqueda local es un método heurístico que permite abordar problemas complejos de optimización y encontrar soluciones que, sin ser óptimas, se aproximan a serlo. Es por esto que las aplicaciones de búsqueda local abarcan campos del conocimiento tan generales como la informática, las matemáticas, la ingeniería o la bioinformática.

En muchos casos, en los problemas de optimización, el conjunto de posibles soluciones está limitado por un conjunto de restricciones. En el diseño de una red de telecomunicaciones para una región geográfica, por ejemplo, en muchos casos el objetivo de minimizar el costo de su implementación debe enfrentar una restricción de distancia originada por el retraso o la pérdida de señal. También existen conjuntos de problemas que requieren una ruta entre un vértice particular de la red y todos los vértices restantes, que no exceda un límite en su longitud y se minimice la cantidad de cable utilizado.

Estos ejemplos son instancias de un problema de complejidad *NP-hard* conocido como problema del árbol de recubrimiento mínimo restringido por la distancia raíz (*Rooted Distance-Constrained Minimum Spanning Tree*, RDCMST). En él, dado un grafo ponderado y un vértice raíz, se debe encontrar un árbol de recubrimiento mínimo en el cual, el camino más corto desde cualquier vértice hasta la raíz no exceda cierto umbral.

Arbeláez et al. [1] presentaron una estrategia para resolver el RDCMST para una aplicación de diseño de redes de comunicaciones ópticas, específicamente de un tipo de red conocida como LR-PON (*Long-Range Passive Optical Network*), que se caracteriza por conectar un conjunto de servidores principales –metro-nodos–, a un conjunto de servidores más pequeños –sitios de intercambio–, que utilizan fibras ópticas cuya longitud está limitada por la pérdida de señal; los sitios de intercambio finalmente se conectan con los clientes finales para formar una red como la que se muestra en la Figura 1. La manera de conectar cada metro-nodo con un grupo de sitios de intercambio o cada sitio de intercambio con un grupo de clientes se puede formular como una instancia de problema RDCMST.

Arbeláez et al. [1] lo formularon como un problema de programación entera mixta (*Mixed Integer Programming*, MIP), una generalización de RDCMST de

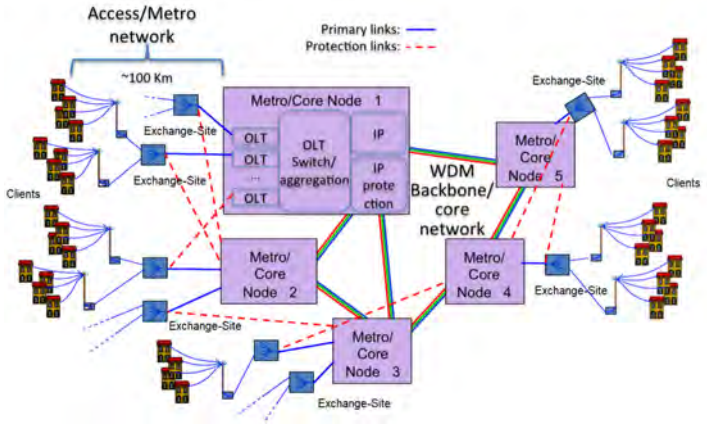


Figura 1. Ejemplo de una red óptica pasiva de largo alcance [1]

aristas disjuntas (*Edge-Disjoint RDCMST*, ERDCMST), esto es, un conjunto de instancias de RDCMST superpuestas con una restricción de aristas disjuntas. También propusieron una búsqueda local iterativa basada en restricciones (*Iterated Constraint-Based Local Search*, ICBLS) capaz de resolver tanto ERDCMST como RDCMST y presentaron una extensión paralela de su algoritmo, solo para ERDCMST, que básicamente puede resolver muchas instancias de RDCMST en paralelo, pero con estrictas limitaciones en el tamaño del grafo. En la literatura, no se encontraron otros enfoques para resolver el problema de RDCMST en paralelo ni de manera distribuida.

En el ICBLS propuesto por Arbeláez et al.[1], el espacio de búsqueda está formado por todos los árboles de recubrimiento posibles que satisfagan las restricciones del problema. Dada una solución subóptima S, se aplica una operación m iterativamente sobre S para construir la vecindad de S en el espacio de búsqueda; en cada iteración de la búsqueda local, el algoritmo elige la solución vecina de S que minimiza la suma de los pesos de todas las aristas del árbol. Esta operación m realiza cambios en la estructura del árbol que van construyendo un árbol que representa la solución aproximada. La exploración de la vecindad en el algoritmo consiste entonces en aplicar el operador m a la solución S, tantas veces como sea necesario, para generar una buena cantidad de posibles soluciones derivadas de S que cumplan con las restricciones del problema. Después de esto, se debe calcular el costo de todas las soluciones posibles y elegir la mejor opción.

El método simplex [2] es un algoritmo que resuelve problemas de programación lineal mediante una búsqueda exhaustiva que garantiza una solución óptima. La ICBLS propuesta por Arbeláez et al. [1] encuentra soluciones aceptables en un tiempo considerablemente más corto que el usado por el método simplex para la formulación MIP –que desborda la memoria incluso en entradas relativamente pequeñas–. Aún así, ICBLS tiene serias limitaciones en cuanto al tamaño del problema de entrada.

Cómo resolver el problema RDCMST mediante una estrategia de exploración de vecindarios en paralelo superando las limitaciones en el tamaño de entrada, es el problema al que se dirige la presente investigación. Resolverlo incluye varios desafíos importantes:

- para maximizar la paralelización de la exploración de vecindarios, se debe definir una granularidad de tarea adecuada, entonces ¿cuál debería ser la que permite obtener el mayor nivel de paralelismo?,
- ¿cómo diseñar una estrategia de distribución que maximice el uso de recursos computacionales y minimice la comunicación entre tareas distribuidas?,
- ¿cuál es el paradigma de computación distribuida (basado en grafos) más conveniente para el diseño e implementación de la estrategia?
- ¿cómo codificar el algoritmo completo de búsqueda local para RDCMST en los modelos de programación soportados en los *frameworks* de computación distribuida basada en grafos del estado del arte, y
- ¿cómo evaluar la solución de una manera comparable con los enfoques descritos en el estado del arte?

Con base en ello, se definió como objetivo general: construir una búsqueda local distribuida con una estrategia de exploración de vecindarios en paralelo basada en una arquitectura de software distribuida para resolver el problema RDCMST, equilibrando la distribución de carga, en la medida de lo posible, de manera uniforme entre los procesadores disponibles habilitados para procesar grandes grafos de entrada [3], y como objetivos específicos: encontrar la granularidad de tarea adecuada para una distribución de carga, lo más uniforme posible, en la ejecución de la búsqueda local distribuida; diseñar un programa distribuido que implemente la estrategia de solución de tal manera que explote los paradigmas de la computación distribuida de grafos; y evaluar, experimentalmente, la corrección y el rendimiento de la solución y compararla con las soluciones de última generación publicadas.

MARCO TEÓRICO Y ESTADO DEL ARTE

ÁRBOLES DE RECUBRIMIENTO MÍNIMO

El árbol de recubrimiento mínimo [4] de un grafo no dirigido ponderado es el árbol que conecta todos los vértices del grafo de manera tal que se minimiza la suma de los pesos de sus aristas. Encontrar ese árbol es un problema ampliamente estudiado por las ciencias de la computación, el cual, desde 1930 se resuelve en tiempo polinomial utilizando algoritmos voraces, y cuya solución hoy se puede encontrar en tiempo lineal o casi lineal. Además, se han propuesto algoritmos paralelos / distribuidos que encuentran una solución al problema con mayor rapidez que los algoritmos secuenciales optimizados.

Han surgido varios problemas del problema del árbol de recubrimiento mínimo, algunos de los cuales son computacionalmente más complejos que el original y requieren estrategias radicalmente diferentes para su solución.

El RDCMST se puede definir así: dado un grafo ponderado $G = (V, E)$ con un nodo raíz $r \in V$, encuentre un árbol de recubrimiento mínimo tal que la distancia de una ruta en el árbol entre cualquier nodo $v \in V$ y la raíz r no exceda un umbral constante de λ. El RDCMST es un reconocido problema en el campo del diseño de redes, conocido por ser *NP-hard* [5], [6], su complejidad puede entenderse fácilmente si se considera una generalización del problema del árbol de recubrimiento mínimo con restricciones de salto, que se muestra como *NP-hard* en [7].

Se han propuesto algunos algoritmos exactos que producen la solución óptima a este problema. Pyo, Oh y Pedram [5] propusieron uno basado en intercambios iterativos de suma negativa, mientras que Gouveia, Paias y Sharma [6] presentaron dos enfoques: uno siguiendo un esquema de generación de columnas, otro mediante una relajación lagrangiana combinada con un procedimiento de optimización de subgradientes. Además, se han modelado y utilizado enfoques clásicos de programación lineal [1], [6]. Sin embargo, estos métodos solo pueden manejar grafos de algunos cientos de nodos.

Para resolver grandes instancias del problema se han introducido varios enfoques heurísticos, entre ellos la búsqueda iterativa local. Algunos de estos métodos fueron diseñados utilizando las ideas de ambiciosos algoritmos clásicos diseñados para resolver MST: el algoritmo de Prim y el algoritmo de Kruskal.

Salama et al. [8] presentaron una heurística basada en Prim que inicia en el nodo raíz; el algoritmo construye el árbol agregando iterativamente el nodo al que se puede llegar de la manera más barata, sin violar la restricción de distancia, cuando no se puede agregar más nodos, se intenta relajar la restricción de distancia cambiando algunas de las aristas de los nodos que ya se han agregado al árbol; si se han agregado todos los nodos, se usa una segunda fase que utiliza el proceso de intercambio de arista para reducir el costo final de la solución.

Pyo, Oh y Pedram [5] presentaron el algoritmo Bounded KRUSkal (BKRUS), el cual, como en su versión clásica, inicia con un grafo desconectado completo y en cada paso elige la mejor arista posible –aquella con el peso mínimo–, que une los componentes de dos grafos, hasta que el grafo queda completamente conectado. En cada paso, el algoritmo es muy cuidadoso de que no haya posibilidad de violar la restricción de distancia en el árbol resultante.

Por su parte, Ruthmair y Raidl [9] presentaron un enfoque basado en Kruskal muy similar. Estos enfoques basados en Kruskal tienden a superar a los basados en Prim en instancias euclidianas porque realizan búsquedas más globales, lo que puede evitar la pérdida de distancia –lo que sí puede suceder en la vecindad de la raíz circundante de una solución basada en Prim–.

Ruthmair y Raidl [10], [11] y [12] han propuesto otros enfoques de *soft computing*. Junto con Berlakovich [10] presentaron una búsqueda local basada en un puntaje de clasificación que organiza las aristas usando su conveniencia potencial de acuerdo con la restricción de distancia, con un enfoque incorpora la restricción de distancia a la heurística, a diferencia de los clásicos que la ignoran mientras no se viole. En [11] propusieron dos métodos: el primero basado en los principios de optimización de colonias de hormigas (*Ant Colony Optimization*, ACO), el cual realiza exploraciones de vecindarios en dos estructuras diferentes; y el segundo que corresponde a una búsqueda de vecindario variable (*Variable Neighborhood Search*, VNS), que usa las mismas estructuras de vecindario, pero introduce intercambios de aristas para perturbar las soluciones; de ellos, el primero, por lo general, muestra mejores resultados. Finalmente, en [12] presentaron un método adicional, muy similar al anterior, pero utilizando un algoritmo genético para realizar la exploración del vecindario. Todos estos métodos han sido validados en grafos de hasta mil vértices.

BÚSQUEDA LOCAL

Búsqueda local (*Local Search*, LS) es un método heurístico que permite encontrar una de muchas posibles soluciones (espacio de búsqueda), mediante pequeñas variaciones de una solución intermedia, siguiendo criterios de optimización o mejora [13]. Usualmente, en una LS, dada una solución S, hay un conjunto de operaciones o movimientos M que al aplicarse a S construyen el vecindario de S, el cual es nada más que un conjunto de soluciones posibles. Un algoritmo de búsqueda local, en cada iteración elige al vecino S que mejora el criterio de optimización y para la siguiente iteración lo usa como su solución intermedia, pero mejorada.

Búsqueda local iterativa

La búsqueda local iterativa (*Iterated Local Search*, ILS) es un mecanismo que permite búsquedas locales para evitar el estancamiento en mínimos locales. Arbeláez et al. [1] presentaron un ILS para resolver el problema. Para evitar dicho estancamiento, se utiliza el Algoritmo 1.

```
1: s* := localSearch(s)
2: repeat
3:     s' := perturbation(s*)
4:     s'' := localSearch(s')
5:     s* := stopCriterion(s*, s'')
6: until Stop criterion reached
```

Algoritmo 1. Búsqueda local iterativa

Dada una solución inicial s, se aplica una búsqueda local alrededor de ella que produce un óptimo local $s*$. A partir de ahí, se realiza un proceso iterativo de tres etapas: primero, se ejecuta una operación sobre la solución $s*$, que produce una solución s' que no corresponde necesariamente a la vecindad de $s*$, a esta operación se le llama perturbación; a continuación, se realiza una búsqueda local alrededor de s', que da como resultado una s'' óptima local; finalmente, una función acceptance decide cuál de los dos óptimos es mejor candidato para ser la solución final del algoritmo o para ser perturbado en una próxima iteración. Por supuesto, el bucle debe terminar de acuerdo con un criterio de parada.

La fase de perturbación ayuda a evitar el estancamiento en mínimos locales porque permite seguir rutas dentro del espacio de búsqueda que serían inaccesibles

o descartadas si se utilizara una simple búsqueda local. Este concepto de perturbación se utiliza en otros dominios, con una amplia gama de aplicaciones, como en los sistemas de software autoadaptativos [14], [15], [16] y [17].

Búsqueda paralela local

Una búsqueda local puede paralelizarse de dos maneras, con métodos *multi-walk* o con métodos de recorrido único. Los métodos *multi-walk* consisten en la ejecución en paralelo de varios algoritmos o varias copias de un mismo algoritmo (utilizando diferentes soluciones iniciales), buscando que cada uno cubra un área diferente del espacio de búsqueda (estos algoritmos pueden funcionar con o sin cooperación entre ellos); los métodos de recorrido único, por su parte, usan la paralelización dentro de un único proceso de búsqueda, una forma de hacerlo es paralelizando la exploración del vecindario.

HADOOP

Hadoop [18] es una colección de herramientas de código abierto útil para almacenar grandes cantidades de datos y ejecutar aplicaciones distribuidas, sobre esos datos, en clústeres que normalmente se construyen con *hardware* básico. Hadoop permite el desarrollo de software de procesamiento de *Big Data* que satisface atributos de calidad, tales como: disponibilidad, escalabilidad, recuperabilidad y concurrencia. Todo esto se logra sin la intervención de los desarrolladores, lo que le permite a ellos enfocarse en la lógica de los algoritmos.

A menudo, el término Hadoop se usa para referirse solo a las herramientas centrales de lo que se conoce como el ecosistema Hadoop, el núcleo compuesto por tres subsistemas: el sistema de archivos distribuido de Hadoop (*Hadoop Distributed File System*, HDFS), el Hadoop YARN (*Yet Another Resouce Negotiator*) y el *Hadoop MapReduce*.

El sistema de archivos distribuido de Hadoop

El HDFS es un sistema de archivos distribuido y escalable diseñado para almacenar conjuntos de datos que superan la capacidad de almacenamiento de una sola máquina. Este sistema de archivos proporciona una interfaz de línea de comandos muy similar a la de Unix. HDFS fue diseñado para ser tolerante a fallas y proporcionar disponibilidad. Todas estas características hacen que la mayoría de las aplicaciones de Hadoop se basen en HDFS como capa de datos.

Tal como en los sistemas de archivos estándar y en los propios discos, HDFS tiene un tamaño de bloque que corresponde a la cantidad mínima de datos que puede leer o escribir. Mientras que el tamaño de bloque de los sistemas de archivos es, por lo general, de unos pocos miles de *kilobytes*, en HDFS este tamaño es mucho mayor. Cabe indicar que un tamaño de bloque de cientos de *megabytes* permite minimizar el costo de realizar búsquedas en discos físicos.

Los bloques, entre otras cosas, son útiles para proveer replicación, lo que garantiza la tolerancia a fallas y la disponibilidad. Cada bloque se replica en un número fijo de máquinas (tres, por defecto). Cuando una máquina falla o un bloque está dañado, se puede leer una copia desde otra ubicación. Asimismo, se genera una nueva copia en una máquina en vivo para mantener constante el número de réplicas. Cuando un cliente solicita un bloque de archivo específico, HDFS se encarga de entregar ese dato desde la máquina más cercana al cliente, que suele ser la misma. Esta característica conduce a la localidad de datos durante la ejecución de aplicaciones Hadoop.

Hadoop YARN

Primordialmente, YARN es la capa del administrador de recursos de Hadoop, aunque también actúa como un programador. Este subsistema proporciona las API para solicitar recursos de clúster, los cuales no son utilizados directamente por los usuarios desarrolladores, sino mediante *frameworks* de procesamiento creados en YARN, como MapReduce o Spark.

Una solicitud de aplicación en Hadoop incluye una cantidad de memoria, una cantidad de núcleos de CPU y, por lo general, una restricción de localidad, para llevar el procesamiento a los datos y no los datos al procesamiento. Si por ejemplo, una aplicación MapReduce necesita leer un bloque HDFS, puede solicitar recursos a YARN en la misma máquina en la que está almacenado el bloque. YARN es también el programador de trabajos cuando no hay suficientes recursos para todas las solicitudes y proporciona algunas estrategias que el usuario puede utilizar para indicar de qué manera deben esperar las aplicaciones o cómo deberán dividirse los recursos en clústeres compartidos.

Hadoop MapReduce

En la primera versión de Hadoop, solo era posible usar una implementación distribuida del modelo de programación MapReduce para procesar datos

almacenados en HDFS. Esta implementación se conoce como Hadoop MapReduce y sobre ella se construyeron muchas otras herramientas de Hadoop. Giraph es uno de esas herramientas (aunque se considera una aplicación MapReduce falsa y algunas de sus nuevas versiones ya no dependen de MapReduce).

En la ejecución de MapReduce interviene un grupo de mapeadores, estos son pequeños programas cada uno encargado de procesar una parte de los datos, para producir un resultado parcial; luego, después de otras fases internas en el *framework*, los reductores resumen o combinan los resultados parciales y producen el resultado final. El usuario desarrollador solo tiene que escribir las funciones mapeadoras y reductoras sin preocuparse de cómo el *framework* paraleliza el cómputo.

GIRAPH

Giraph es un *framework* de computación distribuida que se ejecuta por encima de Apache Hadoop, que le permite a los desarrolladores escribir algoritmos de grafos iterativos que se pueden ejecutar en grafos a gran escala de millones de vértices [19]. A diferencia de los sistemas de bases de datos de grafos, como Neo4 o Thinkerpop, que son útiles para transacciones en línea, Giraph es una herramienta de computación *offline* que generalmente realiza análisis de todo el grafo, una tarea que puede tomar minutos, horas o días. Giraph fue creado como la versión de código abierto de Pregel, una arquitectura de procesamiento de grafos desarrollada en Google [20]. Ambos sistemas utilizan un modelo de programación centrado en vértices basado en el modelo de computación *Bulk Synchronous Parallel* (BSP). Además del modelo de computación básico de Pregel, Giraph ofrece otras características, entre ellas: *master computation*, *sharded aggregators*, *edge-oriented input* y *out-of-core computation*.

Modelo de datos

Un grafo en Giraph está representado por una estructura de datos distribuidos que sigue un enfoque de corte de arista. Es decir, que los vértices del grafo están particionados y distribuidos entre los nodos de procesamiento del clúster. Además, todas las aristas del grafo se asignan a su fuente de vértice; tanto los vértices como las aristas tienen un valor asociado, que puede ser un objeto de cualquier tipo. Un vértice se compone de un ID, un valor y un conjunto de aristas salientes, las cuales, al mismo tiempo, tienen valores asociados y el ID

de sus vértices objetivo. Como resultado, el modelo de datos de Giraph es un grafo dirigido, en el cual los vértices solo tienen acceso directo a sus aristas salientes y luego, si necesitan conocer las entrantes, deben descubrirlas durante la computación de Giraph. Además de los atributos, los vértices se comportan como se ilustra en el Código 1.

```
class Vertex:
        map<int , any> edges #1
        function voteToHalt() #2
        function addEdge(edge) #3
        function removeEdges(targetId) #4
```

Código 1. Clase Vertex

Modelo BSP

Bulk Synchronous Parallel (BSP) es un modelo de computación basado en el paso de mensajes para lograr escalabilidad en la ejecución de algoritmos paralelos en múltiples nodos de procesamiento. En BSP hay N nodos de procesamiento que pueden comunicarse entre sí a través de una red utilizando, por ejemplo, la interfaz de paso de mensajes (MPI, *Message Passing Interface*). La entrada de un algoritmo debe dividirse entre esos nodos de procesamiento y cada uno debe calcular una solución intermedia –o parte de ella–, localmente.

Al finalizar este computo –que se ejecuta en paralelo–, los nodos de procesamiento intercambian sus resultados intermedios a través de mensajes; luego, todos deben esperar hasta que el otro haya terminado. Cuando se han entregado todos los mensajes, comienza una nueva iteración, en ella, cada nodo de procesamiento computa una nueva solución intermedia a partir del estado computado previamente y los mensajes recibidos. En Giraph, la fase de espera se conoce como barrera de sincronización y cada iteración como un superpaso. En la Figura 2 [19] se muestra como en Giraph la partición de entrada se realiza a nivel de vértice y se presenta una imagen aproximada del modelo de computación Giraph.

Computación básica

Como desarrolladores, los usuarios de Giraph no tienen que preocuparse por las complejidades de los sistemas distribuidos del *framework*, prácticamente, solo tienen que lidiar con la función definida por el usuario (UDF, *User Defined*

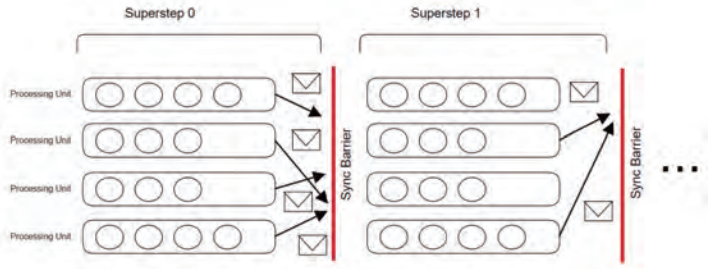


Figura 2. Modelo BSP [19]

Function), llamada función compute, la que se invoca repetidamente para cada vértice y representa el núcleo del modelo centrado en vértices. En otras palabras, un programador Giraph solo tiene que pensar como un vértice, el cual tiene un valor que puede cambiar durante los intercambios de mensajes, algo que ocurre durante muchas iteraciones entre sus vecinos de vértice.

La computación en Giraph se compone de una serie de superpasos que pueden ser vistos como iteraciones de un algoritmo. En cada superpaso, un vértice recibe mensajes de otros vértices, que deben procesar en el siguiente superpaso. Un vértice puede acceder a su valor y a sus aristas, cambiarlos y enviar mensajes a otros vértices; también puede votar para detener la computación.

Cada mensaje enviado se entrega al vértice de destino al comienzo del siguiente superpaso. Si un vértice vota para detenerse, se vuelve inactivo hasta que recibe mensajes más tarde. Un vértice inactivo no se considera en una ejecución de superpaso. La computación finaliza cuando todos los vértices se vuelven inactivos.

Las transiciones entre superpasos están restringidas por una barrera de sincronización, lo que significa que el siguiente superpaso no puede comenzar hasta que se hayan computado todos los vértices activos. Por tanto, la computación en Giraph puede ser considerada sincrónica. En la Figura 3 se muestra la computación de un vértice que recibe dos mensajes, actualiza su valor con el más grande y envía el valor a sus hijos.

Los nodos de procesamiento son responsables de ejecutar la función compute de todos los vértices asociados a ellos; en cada superpaso, deben entregar los mensajes producidos por sus vértices a los nodos de procesamiento que contienen los vértices de destino.

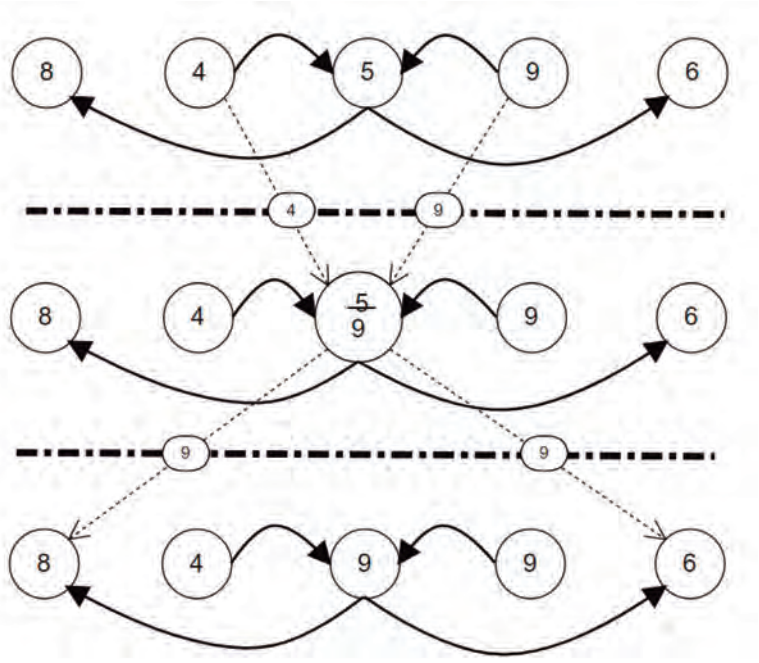


Figura 3. Computación de un vértice que propaga el mayor valor recibido [19]

En cuanto a la API, en la práctica, el programador tiene que implementar el método de computación de una clase llamada BasicComputation. Este método tiene dos parámetros: un vértice y los mensajes enviados a ese vértice desde el superpaso anterior. En cada superpaso, Giraph invoca este método en todos los vértices, entregando sus mensajes correspondientes. Los métodos más relevantes de la computación básica se muestran en el Código 2.

Combiner

Como se puede observar, en lugar de una comunicación de estado compartida, Giraph se basa en el paradigma de comunicación de paso de mensajes. Además, debido a que los mensajes se envían a nivel de vértice, la cantidad de ellos puede ser enorme. Combiner es una función que permite reducir el número de mensajes que se envían a un vértice, con lo que reduce el tráfico de red generado entre los nodos de procesamiento. En la práctica, combiner es solo una función que convierte dos mensajes en uno. Esta función, también llamada Combine Function, se puede ejecutar antes de cada superpaso, pero no hay garantía de cuántas veces será invocada. En la Figura 4 se muestra cómo

```
class BasicComputation:
        function compute(vertex, messages) #1
        function getSuperstep() #2
        function getTotalNumVertices() #3
        function getTotalNumEdges() #4
        function sendMessage(targetId, message) #5
        function sendMessagetoAllEdges(vertex, message) #6
        function addVertexRequest(vertexId) #7
        function removeVertexRequest(vertexId) #8
        function getBroadcast(name) #9
        function reduce(name, value) #10
        function aggregate(name, value) #11
        function getAggregatedValue(name) #12
#1 The method to implement, which is called by the Giraph runtime
#2 Returns the current superstep
#3 Returns the total number of vertices in the graph
#4 Returns the total number of edges in the graph
#5 Sends a message to the target vertex
#6 Sends a message to the endpoints of all the outgoing edges of a vertex
#7 Requests the addition of a vertex to the graph
#8 Requests the removal of a vertex from the graph
#9 Get value broadcasted from master
#10 Reduce given value for a reduce operation
#11 Reduce given value for an aggregator
#12 Get aggregated value of an aggregator
```

Código 2. Clase Basic Computation

se puede reducir el número de mensajes entregados, de tres a uno, llamando dos veces a la función combine.

Arquitectura Giraph

Giraph sigue un patrón maestro-trabajador, en el que los nodos de procesamiento son los trabajadores. Los principales propósitos de esos trabajadores son: ejecutar la función compute y exponer los servicios que permiten la comunicación directa entre trabajadores. Por su parte, el nodo de procesamiento maestro debe: coordinar la transición de los trabajadores entre los superpasos, asignar las particiones de los nodos a los trabajadores, monitorear el estado de los trabajadores y ejecutar el código master compute.

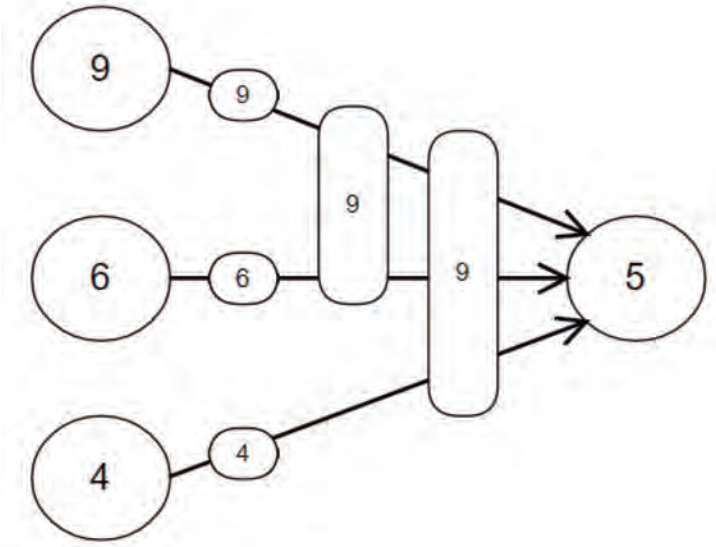


Figura 4. Mensajes reducidos por un combiner Max [19]

Giraph se despliega como una aplicación MapReduce para que pueda ejecutarse en un clúster de Hadoop. A diferencia de las aplicaciones típicas de MapReduce, Giraph está compuesto solo por un número fijo de mapeadores (sin reductores), cuyo propósito es iniciar un servicio que pueda convertirse en trabajador o maestro. Una vez que estos servicios han sido lanzados, quedan completamente a cargo de la computación. Por lo tanto, aunque Giraph está construido sobre MapReduce, su comportamiento no tiene nada que ver con ese modelo de programación. Esa es la razón por la que Giraph se considera un MapReduce falso. En la Figura 5 se muestra un diagrama de alto nivel de la arquitectura de Giraph –las nubes representan particiones de vértices que se asignan a los trabajadores–, y se incluyen los principales métodos para procesar y comunicar datos.

Master compute y shared state

La función master compute es una fase opcional de la computación de Giraph que permite especificar una computación centralizada [21]. El código de esta función se ejecuta secuencialmente en el nodo maestro, antes del inicio de cada superpaso. La función master compute puede cambiar dinámicamente tanto la función compute como la función combine, que se ejecutará en el

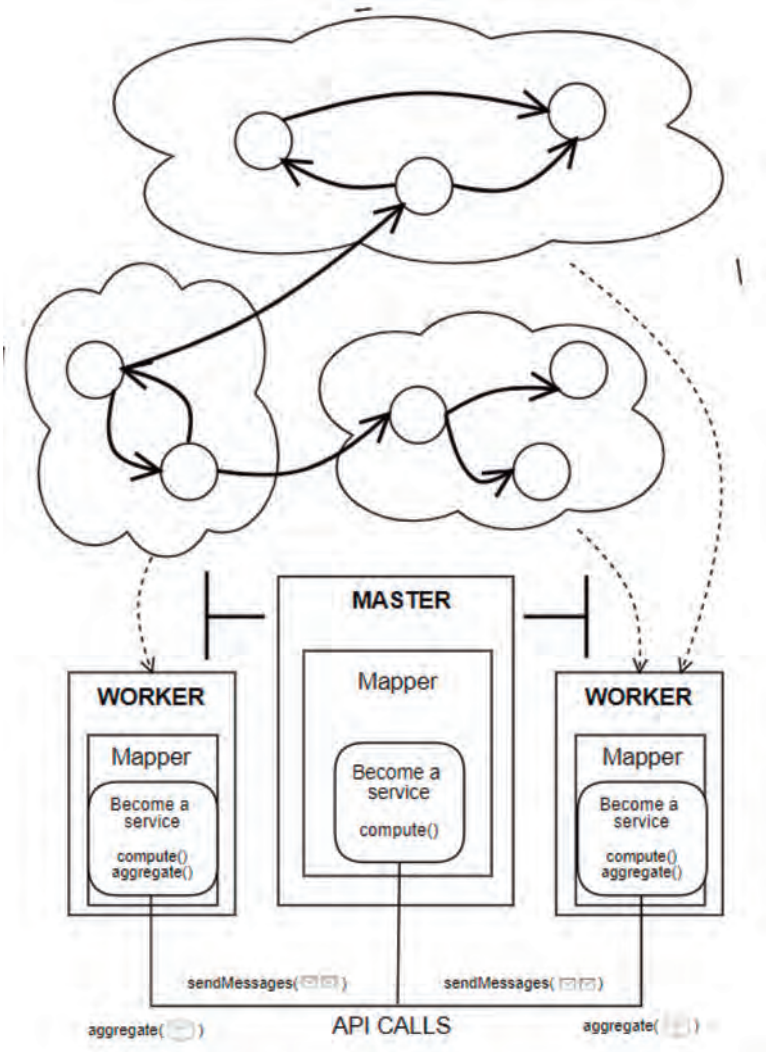


Figura 5. Arquitectura Giraph

siguiente superpaso; además, puede comunicarse con todos los vértices a través de los diferentes tipos de intercambio de datos entre los nodos que se verán a continuación. Hay tres formas de compartir el estado entre los vértices del grafo, todos ellos deben inicializarse en la función master compute con un ID único: broadcast, ReduceOperation y Aggregators.

Broadcast (trasmitir) es la forma más sencilla de compartir datos entre vértices, es transmitir un objeto desde el nodo principal a todos los trabajadores. Los

trabajadores tienen acceso solo de lectura a este objeto y él solo se puede modificar desde la función master compute. Si el objeto no se modifica entre superpasos, no se vuelve a emitir.

ReduceOperation permite a los trabajadores comunicarse con el equipo maestro de una sola forma. Como cada nodo puede enviar un mensaje al maestro, se debe implementar una función reduce. Ella se puede ejecutar en paralelo y permite que la función master compute obtenga un solo valor al final del superpaso (al igual que la función combine). Los trabajadores solo pueden actualizar el valor de ReduceOperation invocando la función reduce. Por lo tanto, solo el maestro tiene acceso de lectura al valor.

Aggregators, por su parte, son funciones globales que permiten la comunicación en todos los nodos de Giraph. Los vértices pueden enviar valores a los aggregators durante un superpaso, y estos valores se agregan mediante una función reduce (llamada aggregate) proporcionada por el usuario, al igual que en ReduceOperation. Después de un superpaso, el maestro tiene acceso de lectura y escritura al valor agregado, mientras que los trabajadores tienen acceso de lectura y la posibilidad de actualizar el valor nuevamente invocando la función aggregate.

Los aggregators son ejecutados en paralelo por cada trabajador cuyos vértices han invocado la función aggregate. El agregado se realiza justo después de que un trabajador ha terminado de ejecutar la función compute de los vértices. Luego, los valores agregados parciales se envían a un trabajador aleatorio, quien concluye el agregado global y envía el valor resultante al nodo maestro, quien lo devuelve a todos los trabajadores. En la Figura 6 [17] se ilustra una ejecución de Giraph usando la función master compute para comunicar datos a través de agregadores. Como se observa en ella, el nodo maestro es un punto de computación centralizado, cuyo método compute () se ejecuta una vez antes de cada superpaso (los valores de aggregator se pasan desde y hacia los vértices).

Respecto de la API, en Giraph, la función master compute se implementa en la clase MasterCompute Varios métodos más relevantes de esta clase se muestran en el Código 3, donde algunos, como registerAggregator, broadcast o getReduced, son las interfaces para lidiar con el estado compartido desde el lado maestro, y otros, como getBroadcast, reduce y *getAggregated*, de la clase BasicComputation (Código 2), cumplen la misma función.

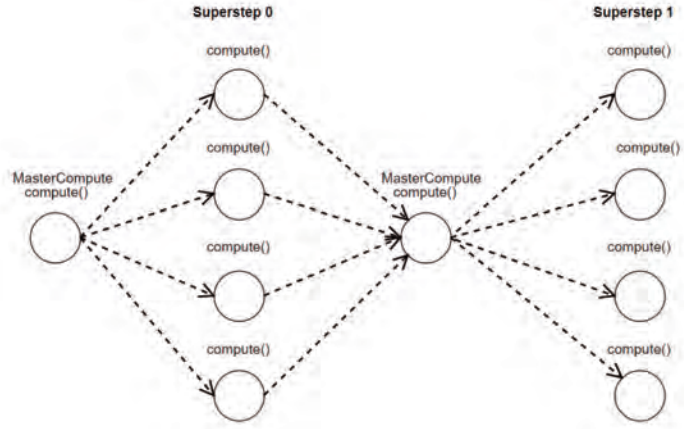


Figura 6. Ejecución de Giraph usando la función master compute **para comunicar datos a través de agregadores [17]**

```
class MasterCompute:
        function compute() #1
        function getSuperstep() #2
        function getTotalNumVertices() #3
        function getTotalNumEdges() #4
        function registerAggregator(name) #5
        function registerReduceOperadion(name) #6
        function broadcast(object) #7
        function haltComputation() #8
        function setComputation(class) #9
        function setCombiner(class) #10
        function getReduced(name) #11
        function getAggregatedValue(name) #12
#1 The method to implement, which is called by the Giraph runtime
#2 Returns the current superstep
#3 Returns the total number of vertices in the graph
#4 Returns the total number of edges in the graph
#5 Register a aggregator
#6 Register a reduce operation
#7 Sends an object to the workers
#8 Halts the computation
#9 Sets the compute function for the next superstep
#10 Sets the combine function for the next superstep
#11 Get reduced value of an reduceOperation
#12 Get aggregated value of an aggregator
```

Código 2. Clase Basic Computation

EL MODELO DE COMPUTACIÓN GIRAPH

Para la mejor comprensión de este modelo de programación y una ilustración de su flujo de ejecución, se presenta un par de subproblemas relacionados con el problema abordado en este proyecto y se describe en detalle, cómo pueden resolverse en Giraph.

UN PROBLEMA SIMPLE

Dado un árbol cuyos vértices almacenan un valor numérico, para cada vértice encuentre el valor más grande del subárbol en el que es la raíz. Al final de la computación, el valor de las hojas debería ser su valor original, y el de la raíz, el valor más grande de todo el árbol. En la Figura 7 se muestra el resultado de la computación dado un grafo de entrada particular.

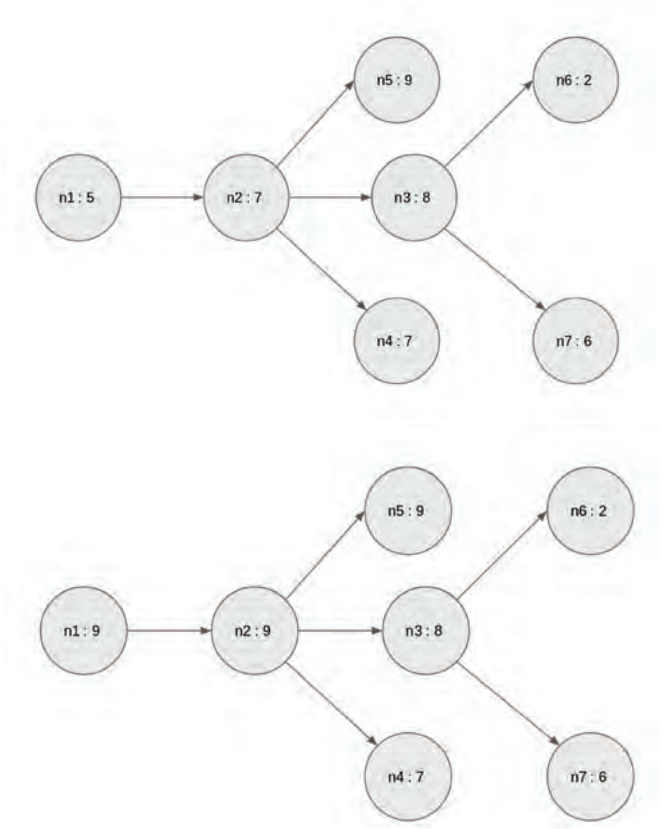


Figura 7. Solución de un problema simple: (a) entradas; (b) solución

Una estrategia para resolver el problema utilizando el paradigma de programación centrada en vértices es la siguiente: para cada vértice, se propaga hacia arriba su valor (es decir, se envía un mensaje a su padre), y se espera hasta cuando llegue un nuevo valor mayor en forma de mensaje. Como consecuencia, el valor del vértice debe actualizarse y propagarse nuevamente hacia arriba. Cuando ya no se envíen mensajes, significa que la computación se ha hecho.

El Algoritmo 2 ilustra la función compute que resuelve el problema usando Giraph. El Algoritmo muestra que se necesita la ID del padre del vértice para hacer la propagación hacia arriba, por lo que se almacena el idPredecessor y el valor numérico del vértice. En el primer superpaso, todos los vértices tienen que propagar sus valores a sus padres, excepto el vértice raíz, porque no tiene uno (líneas 2-5); en los siguientes superpasos, cada vértice tiene que actualizar su valor siempre y cuando uno de sus hijos le envíe un valor mayor que el suyo (líneas 7-9), en ese caso, el nuevo valor debe propagarse (líneas 10-12); finalmente, se invoca la función voteToHalt en la línea 15, lo que permite que la función compute se llame solo cuando hay mensajes, a menos que sea el primer superpaso.

```
1:  subtreeLargestValue.compute = function(vertex, messages)
2:      if getSuperstep == 0 then
3:          if vertex.idPredecessor is not nil then
4:              sendMessage(vertex.idPredecessor, vertex.value)
5:          end if
6:      else
7:          maxValue ← max(messages)
8:          if maxValue > vertex.value then
9:              vertex.value ← maxValue
10:             if vertex.idPredecessor is not nil then
11:                 sendMessage(vertex.idPredecessor, vertex.value)
12:             end if
13:         end if
14:     end if
15:     voteToHalt()
16: end function
```

Algoritmo 2. Longitud promedio de la ruta hasta la hoja más lejana

USO DE LAS FUNCIONES AVANZADAS DE GIRAPH

El segundo subproblema se usa para presentar las características avanzadas de Giraph, en consecuencia, es un poco más complejo. Dado un árbol ponderado, ¿cuál es la longitud promedio del camino hasta la hoja más lejana entre todos

los vértices? Para resolverlo siguiendo el paradigma centrado en vértices, se siguen tres pasos: identificar las hojas del árbol; desde las hojas, propagar las distancias hacia arriba, a través del árbol, hasta que todos los mensajes hayan llegado a la raíz; y, cuando se conozca la distancia de todos los vértices a la hoja más lejana, agregar todos estos valores para calcular su media.

Para diferenciar las etapas del cómputo de Giraph se utiliza una nomenclatura particular para los objetos que se encargan de invocar los algoritmos que se presentan. Cada uno de los nombres representa un entorno de ejecución particular y un momento específico del flujo de ejecución: si el nombre de un objeto termina en computation, eso significa que el objeto es una instancia de BasicComputation y como resultado, ese fragmento de código se ejecuta en paralelo para cada vértice del grafo durante los superpasos; si termina con masterCompute, el código se ejecuta en el nodo principal de forma secuencial; y si termina en aggregator, combiner o reducer, es porque se refiere a un agregador, un combinador o una operación de reducción, respectivamente, por lo que los trabajadores deben ejecutar sus respectivas funciones de agregar, combinar o reducir, justo después de haber calculado todos sus vértices durante el superpaso.

Modelo de datos

Se definió el valor del vértice como un objeto con dos atributos: un número que representa la distancia a la hoja más lejana, llamada distancia hacia atrás o simplemente b; y el *ID* del predecesor del vértice. Cabe recordar que el vértice solo tiene acceso a sus aristas de salida, por lo que, como se requiere propagar mensajes hacia arriba, es necesario que cada vértice conozca el ID de su predecesor único. Solo el vértice raíz no tiene predecesor. Por otro lado, el valor de la arista es un número que representa la distancia entre los nodos.

Master compute

Para comprender la solución macro del problema, se inicia con la función master compute. En el Algoritmo 3 se ilustran, a grandes rasgos, las fases de la computación de Giraph para resolver el problema. Antes de iniciar el primer superpaso (líneas 2-4), se registra un agregador llamado numberVerticesSendingMessagesAggregator, cuyo propósito es indicar al maestro cuando la propagación ha llegado a la raíz, lo que sucede cuando ninguno de los vértices envía mensajes; ahí, el agregador, cuenta el número

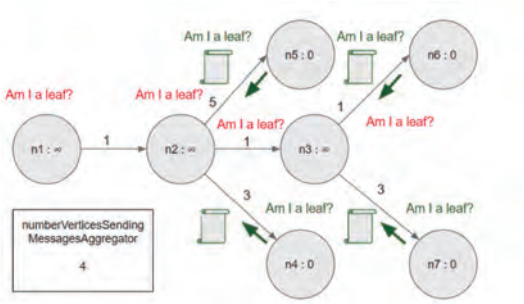
```
1: averagePathLengthToFarthestLeafMasterCompute.compute = function()
2:     if getSuperstep() == 0 then
3:         registerReduceOperation("numberVerticesSendingMessagesAggregator")
4:         setComputation(leafIdentifyComputation)
5:     else
6:         if numberVerticesSendingMessagesAggregator.value is not 0 then
7:             setComputation(propagateComputation)
8:         else if meanAggregator is nil then
9:             registerAggregator("addAggregator")
10:            setComputation(addAggregateComputation)
11:        else
12:            finalResult ← addAggregator.value/getTotalNumVertices()
13:            haltComputation()
14:        end if
15:    end if
16: end function
```

Algoritmo 3. Longitud promedio de la ruta hasta la hoja más lejana

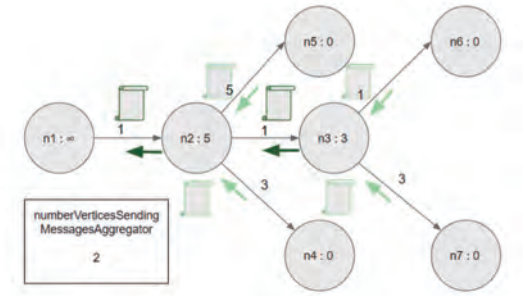
de vértices que envían mensajes en un superpaso específico; posteriormente, se configura la función compute para el primer superpaso (LÍNEA 4), cuyo objetivo es identificar las hojas del árbol. En los siguientes superpasos (LÍNEAS 5-13), mientras que el valor de numberVerticesSendingMessagesAggregator sea diferente de cero, se ejecuta propagateComputation (LÍNEAS 6-8). Esta fase puede verse como el bucle principal del algoritmo, bucle que implica muchos superpasos, durante el cual las distancias se propagan hacia arriba, hasta que todos los mensajes llegan al vértice raíz. Al final del ciclo, cuando todos los mensajes han llegado a la raíz, cada vértice tiene su longitud de ruta hasta la hoja más lejana. Después de esto (LÍNEAS 9-10), se registra un agregador usado para sumar todos los valores *b* (addAggregator) y se establece una nueva función compute para agregar esos valores. En el siguiente superpaso, cada nodo del grafo invoca la función aggregate de addAggregator con las distancias que se han calculado. Al final de ese superpaso, la función aggregate se ejecuta en paralelo para obtener un valor único, el cual se utiliza para computar la solución al problema antes de que se detenga la ejecución (LÍNEAS 12-13).

Computación básica

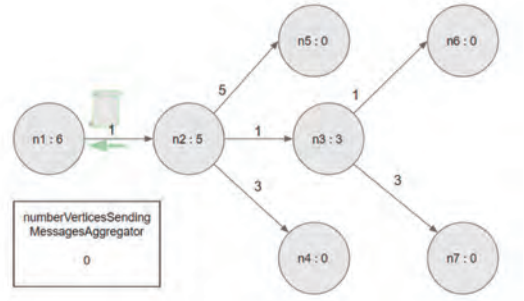
Como se puede observar, se usaron tres funciones compute para resolver el problema. En la FIGURA 8 se muestran estas funciones en acción: el primer superpaso (sección a); propagación de las distancias hasta la hoja más lejana; la primera y la última de las ejecuciones de la función compute de propagateComputation (secciones b y c); y el último superpaso de la totalidad del computo (sección d). Asimismo, con los ALGORITMOS 4, 5 y 6 se explica el detalle de la implementación de cada una de estas funciones.



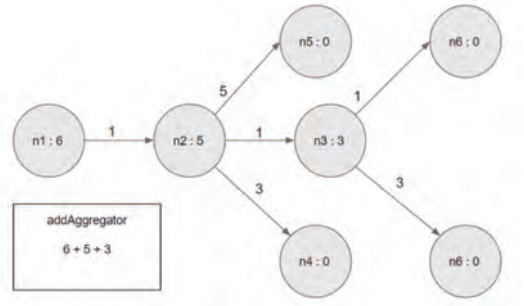
a



b



c



d

Figura 8. Superpasos para identificar la longitud promedio de la ruta hasta la hoja más lejana: identificación de las hojas (a); propagación de las distancias a la hoja más lejana, primera iteración (b) y última iteración (c); y agregado de las distancias finales (d)

```
1: leafIdentifyComputation.compute = function(vertex, msgs)
2:     if vertex.edges.isEmpty() then
3:         vertex.distanceToFarthestLeaf = 0
4:         if vertex.idPredecessor is not nil then
5:             sendMessage(vertex.idPredecessor, (vertex.id, vertex.distanceToFarthestLeaf))
6:             numberVerticesSendingMessagesAggregator.aggregate(1)
7:         end if
8:     end if
9: end function
```

Algoritmo 4. Identificación de las hojas del árbol

```
1: propagateComputation.compute = function(vertex, messages)
2:     if not msg.isEmpty() then
3:         maxPossibleNewB ← 0
4:         for msg in messages do
5:             maxPossibleNewB ← vertex.edges.get(msg[0]) + msg[1]
6:             if maxPossibleNewB > vertex.distanceToFarthestLeaf then
7:                 vertex.distanceToFarthestLeaf ← maxPossibleNewB
8:             end if
9:         end for
10:        if vertex.idPredecessor is not nil then
11:            sendMessage(vertex.idPredecessor, (vertex.id, vertex.distanceToFarthestLeaf))
12:            numberVerticesSendingMessagesAggregator.aggregate(1)
13:        end if
14:    end if
15: end function
```

Algoritmo 5. Propagación de las distancias hacia las hojas más lejanas

```
1: addAggregateComputation.compute = function(vertex, messages)
2:     addAggregator.aggregate(vertex.distanceToFarthestLeaf)
3: end function
```

Algoritmo 6. Agregado de distancias finales

En el Algoritmo 4, un vértice pregunta si su conjunto de aristas salientes está vacío: si es así, el vértice es una hoja y por lo tanto la distancia a su hoja más lejana es cero. Desde este momento, a menos que el vértice sea la raíz, los mensajes comienzan a propagarse hacia arriba y los vértices que envían mensajes comienzan a contarse. Como se observa en este ejemplo, un mensaje es una 2-tupla en la que el primer elemento es el id del vértice fuente y el segundo es el valor que se quiere propagar.

En el Algoritmo 5 se describe la fase de propagación de la computación, ella se ejecuta muchas veces a través de superpasos, hasta que se deja de enviar mensajes. Solo los vértices que reciben mensajes (línea 2) participan en la

computación. Como se observa en las líneas 3 a 9, para cada mensaje recibido se calcula un nuevo valor *b* posible sumando el valor *b* del vértice emisor con la distancia hacia él; luego se elige el valor máximo que podría reemplazar al valor *b* actual del vértice; después de eso, la propagación continúa igual que en la fase anterior (líneas 10-13).

Finalmente, en el Algoritmo 6 se muestra el último superpaso de la computación, el cual simplemente llama a la función aggregate, encargada de sumar todos los valores b. Se utilizó un agregador muy simple, cuya única función es sumar los valores que se le envían (Algoritmo 7).

```
1: addAggregator.aggregate = function(newValue)
2:     value ← value + newValue
3: end function
```

Algoritmo 7. addAggregator

ESTRATEGIA DE DISEÑO GLOBAL

UNA BÚSQUEDA LOCAL ITERADA PARA RESOLVER RDCMST

Como se ha dicho, Arbeláez et al. [1] propusieron un *Iterative Learning Control* (ILC) para resolver el problema de RDCMST; este proyecto de investigación nace de la idea de distribuir esa estrategia para explorar las posibilidades y limitaciones de usar el modelo computacional de Giraph para resolver este tipo de problemas.

El problema

El del árbol de recubrimiento mínimo es uno de esos problemas en los que agregar una pequeña restricción cambia radicalmente su complejidad, pasa de ser relativamente fácil a ser mosntruosamente complejo. En el MST clásico, dado un grafo, el objetivo es encontrar un árbol que contenga todos los vértices, minimizando el peso total de la arista, algo que se puede resolver en tiempo lineal; sin embargo, con solo seleccionar un vértice como raíz y restringir la distancia desde esa raíz a cualquier otro vértice del árbol, el problema se convierte en *NP-hard*. A continuación, se describe el enfoque utilizado por Arbeláez et al. [1] para abordar el problema del RDCMST.

Algoritmo secuencial

Sofisticados enfoques exhaustivos, como el método simplex, difícilmente pueden manejar grafos de cientos de vértices [1], por lo tanto, la alternativa más sencilla es un método heurístico para obtener una solución aproximada al problema. El algoritmo que se presenta en esta sección corresponde a una búsqueda local en la que se mejora paso a paso una solución inicial subóptima. En pocas palabras, dado un árbol que es una solución parcial al problema, la búsqueda local consiste en seleccionar aleatoriamente un vértice de él, quitarlo y reinsertarlo en una mejor ubicación, y repetir esto hasta cuando ningún movimiento mejore el costo total de la solución. Una versión simplificada del algoritmo que realiza esta búsqueda local se puede apreciar en el Algoritmo 8.

```
1: list ← {v_i | v_i ∈ V}
2: while list ≠ ∅ do
3:     v_j ← selectVertexRandomly(list)
4:     if feasibleDelete(v_j, Tree) then
5:         oldLoc ← location(v_j)
6:         delete(v_j, Tree)
7:         cost ← costLoc(oldLoc, v_j)
8:         bestLoc ← oldLoc
9:         for loc in locations(Tree, v_j) do
10:            if feasibleInsert(loc, v_j) then
11:                cost' ← costLoc(loc, v_j)
12:                if cost' < cost then
13:                    cost ← cost'
14:                    bestLoc ← loc
15:                end if
16:            end if
17:        end for
18:        Tree ← insert(Tree, bestLoc, v_j)
19:    end if
20:    if bestLoc ≠ oldLoc then
21:        list ← {v_i | v_i ∈ V}
22:    end if
23:    list ← list − v_j
24: end while
25: return Tree
```

Algoritmo 8. LC para RDCMST

Antes de comenzar con la explicación del algoritmo, se debe precisar que se mantiene una matriz de pesos para calcular el costo de una arista entre cualquier par de vértices. En el caso de una instancia euclidiana, por ejemplo, la matriz representaría las distancias euclidianas entre los puntos asignados como vértices; una arista entre dos vértices *vj* y *vk* se denota por *(vj, vk)*.

En la primera línea del algoritmo, se define una variable como la lista de todos los vértices del grafo, donde se almacenan los vértices potenciales que

podrían mejorar la solución actual cuando se muevan, por lo tanto, cuando la lista está vacía, se alcanza un mínimo local y finaliza la búsqueda local, lo que significa que ningún movimiento en el ese árbol puede mejorar la solución (LÍNEA 2). En cada iteración: se selecciona un vértice de la lista, al principio, y se elimina, al final; ese vértice *vj* es el candidato a mover, el primer paso para hacerlo es verificar la viabilidad de la operación de eliminación (LÍNEAS 3-4).

La operación de eliminación implica tanto la eliminación de las aristas entre *vj* y su padre y sus hijos como la reconexión del árbol mediante la creación de aristas entre el antiguo padre de *vj* y los antiguos hijos de *vj*. En la FIGURA 9 se ilustra la operación de eliminación que puede producir un árbol inviable después de realizarse. La función factibleDelete comprueba que no se infrinja la restricción de distancia para continuar con el movimiento; si lo hace, se cancela.

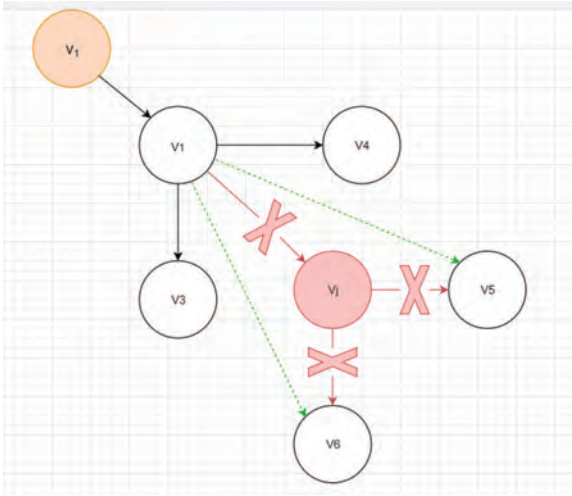


Figura 9. Operación de eliminación

La ubicación de un vértice en el árbol puede verse como una tupla *(vpj, Sj)* donde *vpj* es el padre de *vj* y *Sj* son los hijos de *vj*. Luego, inmediatamente antes de que se realice la operación de eliminación, la ubicación actual de *vj* se guarda en una variable (LÍNEA 5). Por su parte, la función costLoc, que calcula el costo total del árbol como si se insertara un vértice en una ubicación específica, calcula el costo de la solución actual y lo almacena en la variable de costo. Asimismo, la variable bestLoc, que almacena la mejor ubicación para *vj*, se inicializa con su ubicación anterior en la LÍNEA 7.

Un vértice se puede insertar como una nueva hoja hijo para otro vértice o en medio de una arista existente en el árbol (Figura 10): el primer escenario solo implica crear una arista entre cualquier padre nuevo de vj (*vnewpj*) y él mismo *(vnewpj , vj)*; el segundo se logra quitando la arista entre dos vértices *(vm, vn)* y creando las aristas *(vm, vj)* y *(vj, vn)*. En consecuencia, la función locations devuelve todas las ubicaciones posibles en el árbol para el vértice *vj* siguiendo los escenarios mencionados, luego, se evalúa cada ubicación buscando la menos costosa y verificando su factibilidad (líneas 9-17).

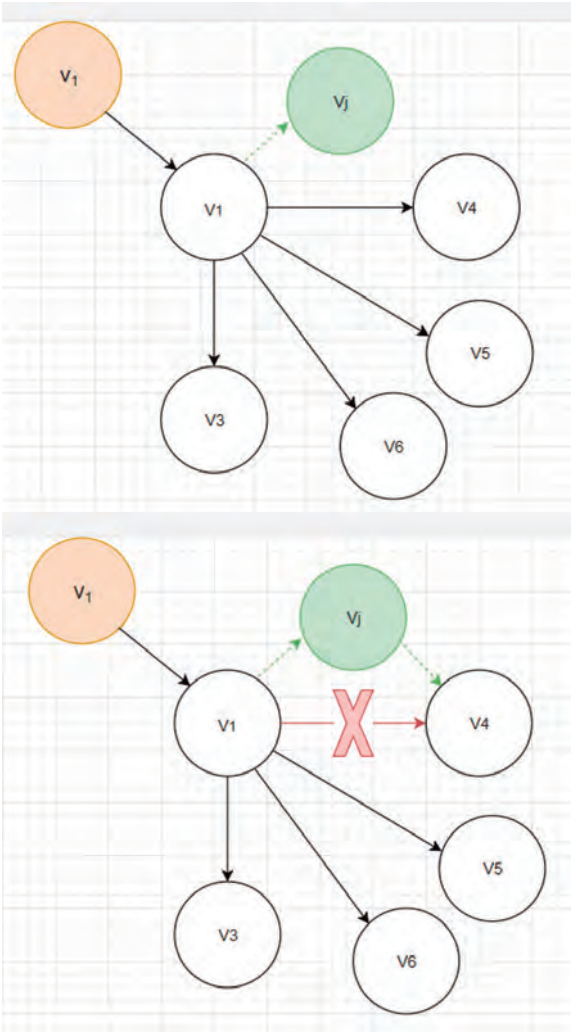


Figura 10. Escenarios para la operación de inserción: como una nueva hoja hijo para otro vértice (a); en medio de una arista existente (b)

Una vez que se han considerado todas las ubicaciones, se realiza la operación de inserción con la mejor ubicación. Además, si se completó un movimiento, todos los vértices se reintroducen en la lista porque con las condiciones de la nueva solución, sus movimientos ahora podrían producir mejoras (LÍNEAS 20-22).

La complejidad del tiempo de un movimiento es $O(n)$, y siempre está determinada por la operación de mejor ubicación, la cual tiene que iterar sobre todos los vértices para comprobar todas las ubicaciones posibles. Las otras operaciones –como las de inserción, eliminación o verificación de la viabilidad–, en el peor de los casos, tienen que pasar por el conjunto completo de vértices, pero generalmente, dependiendo de la implementación, su complejidad es menor.

El ALGORITMO 8 está "arropado" en la plantilla de un ILC general ilustrada en el ALGORITMO 1. En este caso, la función perturbation realiza una serie de movimientos aleatorios sin la mejor heurística de ubicación, lo que significa que solo verifica la viabilidad de la salida.

HACIA UNA SOLUCIÓN PARALELA

Como se mencionó, en la revisión de la literatura no se encontró una solución paralela para el problema de RDCMST; sin embargo, dado que hoy en día el *hardware* de las computadoras es naturalmente paralelo [22], se necesitan algoritmos paralelos para explotarlo. Además, Big Data es una realidad [18] y por ello el diseño de algoritmos distribuidos es una necesidad.

Como se mencionó en la presentación del estado del arte, los enfoques para resolver el problema solo tratan con cientos de vértices o con pocos miles, en el mejor de los casos. En experimentos realizados en casos reales del problema, como el de Arbeláez et al. [1], se alcanzaron diez mil vértices al resolver el problema más general del ERDCMS, que incluye la resolución de RDCMST. El principal objetivo de este proyecto, en consecuencia, es presentar una estrategia que permita manejar instancias mucho más grandes.

El enfoque más natural para paralelizar el ALGORITMO 8 es dividir la exploración de vecindarios en tareas independientes, cada una a cargo de evaluar diferentes ubicaciones. En código, esto significa usar un parallel for en la LÍNEA 9 del algoritmo. Sin embargo, otras operaciones dentro de la búsqueda local, como verificar la viabilidad, pueden aprovechar la paralelización. En la próxima sección se propone una estrategia para paralelizar casi por completo el

Algoritmo 8 en un sistema distribuido siguiendo el paradigma de computación de Pregel [20], en lo que se constituye como el mayor aporte de este proyecto.

SOLUCIÓN COMPLETA BASADA EN GIRAPH PARA RESOLVER RDCMST

Se ha denominado "movimiento" a una iteración completa del Algoritmo 8, un movimiento puede verse como una secuencia de dos operaciones principales, eliminación e inserción: en la primera se selecciona un vértice y se elimina del árbol; en la segunda, el algoritmo elige la mejor ubicación para el vértice eliminado y lo reinserta ahí.

Se implementó un movimiento en Giraph como una secuencia de cinco superpasos, en general, los dos primeros corresponden a la eliminación y los restantes a la inserción. La repetición recurrente de estos cinco superpasos constituye la búsqueda local iterativa para resolver el problema. Aunque esta correspondencia puede parecer bastante sencilla, la distribución entre los superpasos de las operaciones secundarias encargadas de abordar los requisitos previos y las consecuencias de las operaciones principales, en términos del estado de los vértices, es un poco más compleja.

La presentación de la estrategia está estructurada en dos apartados, uno para eliminación, otro para reinserción, cada uno con una subsección en donde se explican las operaciones secundarias. A continuación, se presenta: el modelo de datos de la solución diseñada; una descripción general de la solución, a través de los detalles de la implementación de la función `master compute`; y el diseño de un movimiento completo utilizando Giraph. Como complemento, en el anexo se presenta una descripción gráfica de un movimiento completo, incluyendo todos sus escenarios.

Modelo de datos

Se definió el valor del vértice como una estructura de datos implementada en la clase `VertexValue` (Código 4), sus campos se presentan a continuación. Los ejemplos usados en cada campo se explican considerando la solución parcial de la instancia del problema que se presenta en la Figura 11.

- `double f` es la distancia desde este vértice al vértice raíz en la solución parcial, el valor f del vértice 2 es 3 y el del vértice 8 es 12;
- `double b` es la distancia desde este vértice hasta la hoja más lejana, el valor b del vértice $n3$ es 6 y el del vértice $n8$ es 0;

```
class VertexValue:
        double f #1
        double b #2
        map<int, enum> positions #3
        map<int, double> distances #4
        int parentId #5
        double oldF #6
        double oldB #7
        double partialBestLocationCost #8
        int idSuccToSelectedNode() #9
```

Código 4. Clase MasterComputation

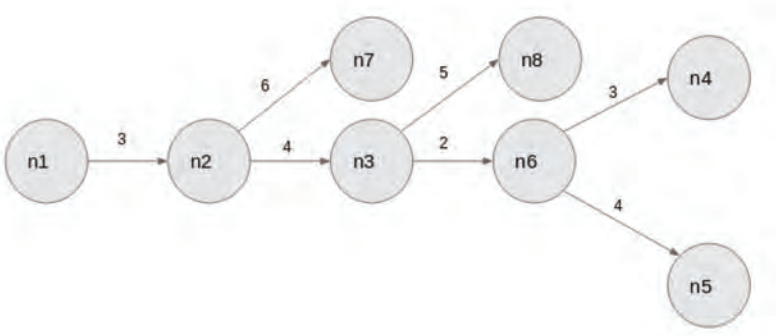


Figura 11. Solución parcial al problema RDCMST

- map <int, enum> posiciones relaciona los vértices del árbol con la posición de este vértice respecto de él, los índices corresponden a los ID de todos los vértices del grafo y los valores son Enums con tres posibilidades que indican, si para el vértice que le corresponde al índice se trata de un ancestro, de un descendiente o de ninguno de ellos (*e.g.*, el mapa del vértice 3 se vería como la estructura del Código 5);
- mapa <int, double> distancias relaciona los vértices del árbol con las distancias de este vértice a ellos, en este mapa, los índices son los mismos de posiciones, pero los valores muestran la distancia que habría si una arista conectará directamente este vértice al vértice correspondiente en el índice;

```
1 : Descendant ,
2 : Descendant ,
3 : None,
4 : Ancestor ,
5 : Ancestor ,
6 : Ancestor ,
7 : None,
8 : None
```

Código 5. Posiciones de vértice de n3

- int parentId es el ID del padre único de este vértice, el parentId del vértice *n3* es 2, mientras que el parentId del vértice n4 es 6 (el único vértice que no tiene un padre es el vértice facility);
- double olfF es el valor de *f* justo después de completar el movimiento anterior;
- double oldB es el valor de *b* justo después de completar el movimiento anterior; y
- double parcialBestLocationCost almacena temporalmente el costo de insertar el vértice seleccionado como hijo de este vértice.

Master compute

El Algoritmo 9 muestra el código master compute: primero se define el criterio de parada de la computación, que es un límite superior para el número de movimientos realizados (línea 1); un movimiento toma cinco superpasos (línea 2) denominados fases de un movimiento, para determinar la fase de un movimiento en particular, se usa la variable superstepStepPhase (línea 6). La función master compute ilustra estas fases estructuradas a través de una declaración de conmutación en donde cada caso corresponde a una de ellas.

Como se dijo, las fases 0 y 1 tratan principalmente de la operación de borrado, y las fases 2, 3 y 4 de la operación de inserción. Aunque la responsabilidad principal de los casos del *switch* es establecer las diferentes funciones de cómputo para los superpasos (líneas 10, 15, 28, 31 y 39), también se encargan de algunos cómputos secuenciales críticos, la mayoría de ellos útiles para actualizar el estado de los vértices después de las operaciones primarias. En el caso 0 (línea 10), por ejemplo, se invoca el método selectANodeMasterCompute (Algoritmo 10), el cual selecciona aleatoriamente un vértice convirtiéndolo en

```
1: maxIterations ← 100
2: superStepsPerIteration ← 5
3: iteration ← 0
4: superstepDeviation ← 0
5: RDCMSTMasterCompute.compute = function()
6:     superStepPhase ← (getSuperstep() + superstepDeviation) mod superStepsPerIteration
7:     if iteration < maxIterations then
8:         switch superStepPhase do
9:             case 0
10:                selectANodeMasterCompute()
11:            case 1
12:                computeBValues()
13:                createGlobalToUpdatePredecessorB()
14:                registerAggregator("parentFAggregator")
15:                setComputation(insertEdgesForDeleteOperationComputation)
16:            case 2
17:                bestPossibleNewBDirPred ← max(PossibleNewBsDirPred))
18:                if parentFAggregator.value + bestPossibleNewBDirPred > lambda then
19:                    superstepDeviation +← 4
20:                    selectANodeMasterCompute()
21:                    break
22:                end if
23:                registerAggregator("parentBestPossibleNewBAggregator")
24:                parentBestPossibleNewBAggregator.value ← bestPossibleNewBDirPred
25:                registerAggregator("AllPredecessorsPossibleNewBsAggregator")
26:                registerAggregator("parentBAggregator")
27:                registerAggregator("selectedNodeChildrenAggregator")
28:                setComputation(updateNodesAndBeginBestLocationComputation)
29:            case 3
30:                computeBValues()
31:                setComputation(finishBestLocationComputation)
32:            case 4
33:                registerAggregator("frustratedMovement")
34:                movementCostAggregator.value +← bestLocation.cost
35:                frustratedMovement.value ← movementCostAggregator.value > 0
36:                registerAggregator("parentBestPossibleNewBAggregator")
37:                registerAggregator("AllPredecessorsPossibleNewBsAggregator")
38:                registerAggregator("parentBAggregator")
39:                setComputation(insertOperationAndUpdateNodesComputation)
40:        iteration++
41:    else
42:        haltComputation()
43:    end if
44: end function
```

Algoritmo 9. Giraph para resolver RDCMST (una solución completa)

```
1: selectANodeMasterCompute = function()
2:     selectedNode ← selectRamdomlyANode()
3:     broadcast("selectedNodeAggregator", selectedNode)
4:     registerReducer("deleteCostForSuccessorsReduceOperation")
5:     registerAggregator("movementCostAggregator")
6:     setComputation(removeEdgesForDeleteOperationComputation)
7: end function
```

Algoritmo 10. Selección de un vértice (master compute 0)

una variable global. Además, el método crea un par de variables relacionadas con la necesaria actualización posterior a la operación de eliminación. Durante la descripción de cada uno de los superpasos (a continuación), se ve en detalle la totalidad de los casos de la función master compute.

Operación de eliminación

Eliminación factible

Previo a una operación de eliminación es necesario verificar si esta operación satisface la restricción de distancia o no. Si la operación de eliminación no es factible, se debe seleccionar otro vértice del grafo a eliminar. Se pueden utilizar las variables b y f de un valor de vértice para comprobar la viabilidad de todos los caminos que cruzan ese vértice, esto se puede hacer directamente, agregando los valores f y b de cualquier vértice y verificando si el resultado es menor que el valor de restricción de distancia.

Para determinar la viabilidad de la operación de eliminación es posible enfocarse únicamente en el padre del vértice seleccionado. Este vértice puede verse como un dominador, porque todos los nuevos caminos que se crean mediante la operación de eliminación tienen que pasar por él, por lo tanto, dado que el valor de f no cambia a causa de la operación de eliminación, si se computa la distancia a la hoja más lejana siguiendo esos nuevos caminos creados, se puede realizar la operación de eliminación factible. En la Figura 12 se ilustra esta estrategia.

Luego de la selección del vértice, en la función master compute, en la fase 0, se agregan los identificadores de los hijos del vértice seleccionado, como las

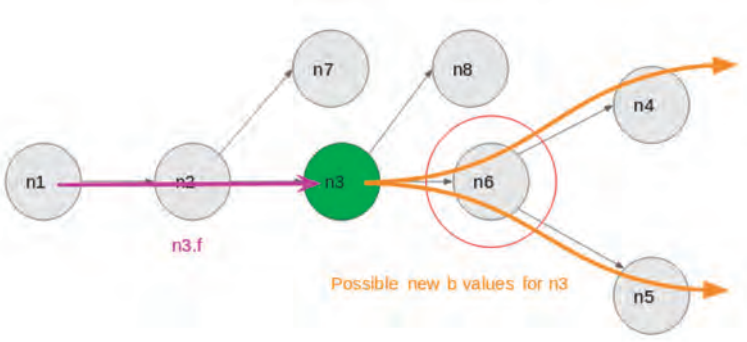


Figura 12. Estrategia de eliminación factible

claves de un agregador llamado succesorsDeleteCostsAggregator, cuyo valor es un mapa (Algoritmo 11, líneas 34-40). Para efectos de la operación de eliminación factible, solo se necesitan esas claves, aunque el agregador realiza otra función.

Estos identificadores permiten identificar las nuevas rutas que se crearían debido a la operación de eliminación, en el caso1 de la función master compute, se crea un nuevo agregador llamado parentPossibleNewBsAggregator, que es

```
1: removeEdgesForDeleteOperationComputation.compute = function(vertex, msgs)
2:     if getSuperstep! = 0 then
3:         if vertex.id == oldSelectedNode.id then
4:             for m in msgs do
5:                 if m.key is a Number then
6:                     vertex.positions.get(m.key) = m.value
7:                 end if
8:             end for
9:             vertex.f ← msgs.get("F")
10:            if bestLocation.way == "FROM_NODE" then
11:                vertex.b ← 0
12:            else if bestLocation.way == "BREAKING_EDGE" then
13:                vertex.b ← msgs.get("BEST_LOCATION_B") + vertex.distances.get(bestLocation.id)
14:            end if
15:        else if vertex.isPredecessor(selectedNode) then
16:            maxB ← msgs.findAll("PAR_B").map(m => m.value + vertex.distances.get(m.key))
17:            if vertex.id == selectedNode.predecessorId then
18:                if bestLocation.way == BREAKING_EDGE then
19:                    if maxB > parentBestPossibleNewBAggregator.value then
20:                        vertex.b ← maxB
21:                    else
22:                        vertex.b ← parentBestPossibleNewBAggregator.value
23:                    end if
24:                end if
25:                parentBAggregator.aggregate((vertex.idParent, vertex.b)
26:            else
27:                possibleNewBUtilities.idParent ← vertex.idParent
28:                possibleNewBUtilities.possNewB ← maxB
29:                possibleNewBUtilities.partialPossNewB ← vertex.distances.get(msgs.get("ID"))
30:                AllPredecessorsPossibleNewBsAggregator.aggregate((vertex.id, possibleNewBUtilities))
31:            end if
32:        end if
33:    end if
34:    if vertex.id == selectedNode.id then
35:        cost ← 0
36:        for edge in vertex.edges do
37:            deleteCostForSuccessors.add(edge.targetId, -vertex.distances.get(edge.targetId))
38:            cost ← vertex.distances.get(edge.targetId)
39:        end for
40:        successorsDeleteCostsAggregator.reduce(deleteCostForSuccessors)
41:    else if vertex.positions.get(selectedNode.id) == PREDECESSOR then
42:        for edge in vertex.edges do
43:            vertex.sendMessage(edge.target, vertex.distances.get(edge.target))
44:        end for
45:    end if
46: end function
```

Algoritmo 11. Remoción de aristas para operación de eliminación (superpaso 0)

```
1: procedure INSERTEDGESFORDELETEOPERATIONCOMPUTATION(vertex, msg)
2:     if getSuperstep! = 0 then
3:         if vertex.positions.get(oldSelectedNode.id) = PREDECESSOR then
4:             if not vertex == selectedNode.parent then
5:                 vertex.b = newBsAggregator.get(vertex.id).value
6:             end if
7:         end if
8:         vertex.oldB ← vertex.b
9:         vertex.oldB ← vertex.f
10:    end if
11:    possibleNewBsDirPred ← parentPossibleNewBsAggregator.value
12:    if vertex.isPredecessor(selectedNode) then
13:        if vertex.idParent is not nil then
14:            vertex.sendMessage(vertex.idParent, ("ID", vertex.Id))
15:        end if
16:        if vertex.id == selectedNode.predecessorId then
17:            deleteCostForSuccessors ← successorsDeleteCostsAggregator.value
18:            cost ← 0
19:            distToSelectedNode ← vertex.distances.get(selectedNode.id)
20:            for key in deleteCostForSuccessors.keySet do
21:                deleteCostForSuccessors(key) +← vertex.distances.get(key) − distToSelectedNode
22:                possibleNewBsDirPred(key) +← vertex.distances.get(key)
23:                cost +← vertex.distances.get(key)
24:            end for
25:            successorsDeleteCostsAggregator.aggregate(deleteCostForSuccessors)
26:            parentPossibleNewBsAggregator.aggregate(possibleNewBsDirPred)
27:            parentFAggregator.aggregate(vertex.f)
28:            movementCostAggregator.aggregate(cost)
29:        end if
30:    else if vertex.isDirectSuccessor(selectedNode) then
31:        possibleNewBsDirPred(vertex.id) ← vertex.b
32:        parentPossibleNewBsAggregator.aggregate(possibleNewBsDirPred)
33:    else if not msg.empty() then
34:        vertex.sendMessage(vertex.predecessorId, ("POSSB", msg + vertex.b))
35:    end if
36: end procedure
```

Algoritmo 12. Inserción de aristas para operación de eliminación (superpaso 1)

```
1: RDCMSTMasterCompute.createGlobalToUpdatePredecessorB = function()
2:     sumDeleteCostForSuccessorsAggregator.value ← successorsDeleteCostsAggregator.value
3:     for key in sumDeleteCostForSuccesorsAggregator.value.keySet do
4:         parentPossibleNewBsAggregator.value(key) ← 0
5:     end for
6: end function
```

Algoritmo 13. Creación de aggregators para computar el costo de eliminación (master compute 1)

```
1: procedure SUMDELETECOSTFORSUCCESORSAGGREGATION(newValue)
2:     for key in newValue.keySet do
3:         sumDeleteCostForSuccesorsAggregator.value(key) +← newValue.get(key)
4:     end for
5: end procedure
```

Algoritmo 14. Suma de Aggregation sobre el costo de la operación de eliminación

un mapa que comparte las mismas claves de succesorsDeleteCostsAggregator (Algoritmo 9, línea 13; Algoritmo 13) y la misma función aggregate (Algoritmo 14). El valor de cada elemento de este agregador es la distancia desde el padre del vértice seleccionado, hasta la hoja más lejana del subárbol cuya raíz es la clave del elemento. Por lo tanto, en la fase 1, el padre del vértice seleccionado debe proporcionarle a parentPossibleNewBsAggregator el costo de conectarlo con todos los hijos del vértice seleccionado (Algoritmo 12, líneas 16-25).

Al mismo tiempo, los hijos del vértice seleccionado deben agregar sus valores b en el mismo agregador en la clave correspondiente (Algoritmo 12, líneas 30-32). Como resultado, al final de la fase, parentPossibleNewBsAggregator contiene las distancias a las hojas más lejanas, desde el padre del vértice seleccionado, siguiendo las rutas que atraviesa cada uno de los hijos del vértice seleccionado. Esos son los posibles valores b del padre del vértice seleccionado que efectivamente podrían romper la restricción de distancia.

Además, en la fase 1 el padre del vértice seleccionado almacena su valor f en un agregador (Algoritmo 12, línea 27). Esto se hace para realizar la operación de eliminación factible en la próxima invocación de la función master compute, para así no tener que esperar hasta el siguiente superpaso. Por lo tanto, en el caso 2 de la función master compute, ya se puede calcular la operación de eliminación factible, primero hay que encontrar el valor máximo de parentPossibleNewBsAggregator y luego verificar si ese valor más el valor de parentFAggregator es mayor que el valor de restricción de distancia; si esto es así, se deben omitir las siguientes fases del movimiento actual y en el siguiente superpaso comenzar un nuevo movimiento completo (algoritmo 9, líneas 17-21).

Consecuencias de la operación de eliminación en el estado del grafo

La operación de eliminación tiene un impacto sobre el estado del grafo, el cambio más notable es topológico e implica la adición y eliminación de aristas. Sin embargo, ese no es el único cambio. Se mantienen algunas variables cruciales en cada vértice que se deben actualizar: las variables f y b, que permiten comprobar fácilmente la viabilidad de la restricción de distancia; y la matriz de posición, que permite a los vértices conocer su posición respecto de cualquier otro vértice, para poder realizar tareas particulares que dependen de ello. Esas variables se deben actualizar después de las operaciones de eliminación e inserción, lo que se logra con varios superpasos, que van desde la primera hasta la cuarta fase, en la que se actualiza la última de las variables.

Algunas piezas del proceso que se presentan en esta sección se comparten con la operación de inserción.

En cuanto al cambio topológico, cabe mencionar que inmediatamente después de que la eliminación factible ha sido verificada y aprobada en el caso 2 de master compute 2, durante el superpaso 2, el vértice seleccionado debe eliminar todas sus aristas (algoritmo 16, líneas 2-8) y el padre del vértice seleccionado debe eliminar su arista al vértice seleccionado y hacer una solicitud para crear aristas desde él, hacia todos los hijos del vértice seleccionado, cuyos identificadores están en ese punto almacenados en sucesoresDeleteCostAggregator (Algoritmo 16, líneas 10-16).

Después de una operación de eliminación, los valores *f* de los descendientes del vértice seleccionado deben actualizarse, porque son los únicos vértices cuyas rutas hacia la raíz se ven afectadas por la eliminación; sin embargo, no es necesario abordar individualmente cada descendiente de vértice seleccionado, todos los vértices de la rama, a partir del hijo de cada vértice seleccionado, pueden actualizar su variable *f* con el mismo valor.

En la Figura 13 se ilustran las diferentes ramas de los descendientes del vértice seleccionado; en este ejemplo, el vértice *n3* es el vértice seleccionado y sus descendientes, que son alcanzados por una línea puntiaguda del mismo color, tienen que actualizar su variable *f* usando la misma variación –o incremento–, que le corresponde al costo de quitar e insertar aristas para la operación de eliminación en esa rama en particular. En el caso de la rama verde, por ejemplo, este costo es *n2.dists[n8] – n2.dists[n3] – n3.dists[n8]*. En este ejemplo, se necesita calcular tres valores, los que se utilizarán para actualizar la variable *f* de todos los descendientes del vértice seleccionado.

Para lograr lo anterior en Giraph, se define un agregador llamado succesorsDeleteCostsAggregator cuyo valor es un mapa en el que cada elemento corresponde a una rama que inicia en el vértice seleccionado. Al final del computo, para cada rama se debe almacenar el valor que se debe sumar al valor *f* actual de todos los vértices de esa rama, para actualizarlos. Este mismo agregador se usó en la operación de eliminación factible, donde se utilizó el conjunto de claves del mapa, que no es más que los identificadores de los hijos del vértice seleccionado.

La función aggregate de este agregador se presenta en el Algoritmo 14, en él, dados dos mapas, básicamente agrega los valores de las claves coincidentes

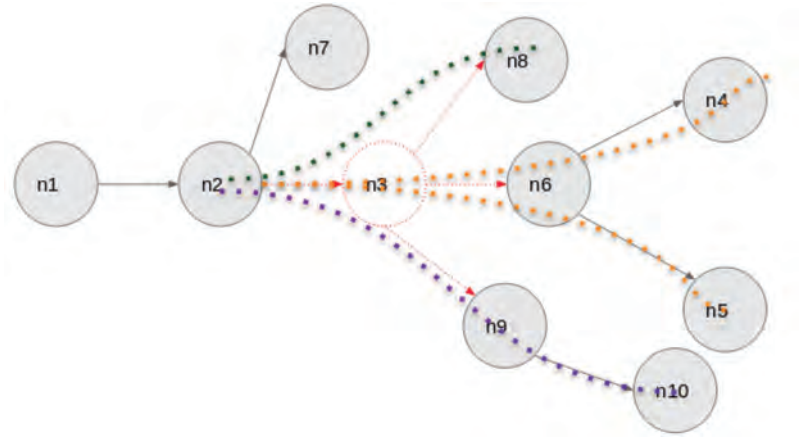


Figura 13. Actualización de los atributos f después de la eliminación de un vértice

y, si hay un elemento en un mapa que no coincide, lo agrega al mapa resultante. El proceso para obtener su valor final a través de superpasos, se explica a continuación.

Primero, en el superpaso 0, el vértice seleccionado crea un mapa con las ID de su hijos, como sus claves, y la distancia a cada hijo en negativo, como su valor. Luego, el mapa se pasa a la función aggregate de successorsDeleteCostsAggregator (Algoritmo 11, líneas 34-40); al final del superpaso, successorsDeleteCostsAggregator tiene el costo positivo de eliminar las aristas del vértice seleccionado para todos sus hijos y solo debe retener los costos de eliminar la arista entre el vértice seleccionado y su padre y el de insertar todas las aristas necesarias para volver a conectar el grafo.

En el superpaso 1, el padre del vértice seleccionado crea un mapa con el mismo conjunto de claves, pero con diferentes valores; para cada rama incluye el costo de conectarse con el respectivo hijo del vértice seleccionado menos el costo positivo de eliminar la arista entre él y el vértice seleccionado (Algoritmo 12, líneas 16-25). Al final, toda la información necesaria para hacer la actualización de los valores de *f* está lista en el agregador y solo se debe esperar al siguiente superpaso para hacerlo.

En el superpaso 2, el descendiente de cada vértice seleccionado tiene que comprobar qué rama le corresponde, utilizando sus vectores de posición y las claves de successorsDeleteCostAggregator; posteriormente, debe actualizar el

valor *f* agregando el valor correspondiente al valor *f* actual (Algoritmo 16, líneas 29-32).

En cuanto a los valores *b*, su actualización solo es necesaria para los antepasados de vértices seleccionados, ya que solo los caminos a sus hojas se pueden ver afectados durante la operación de eliminación. Sin embargo, a diferencia del proceso de actualización de los valores *f*, en el que cada vértice tiene una ruta única a la raíz, para cada vértice hay varias ramas que conducen a muchas hojas potencialmente más lejanas; además, no todos los caminos a las hojas más lejanas de los antepasados del vértice seleccionado conducen al mismo vértice de la hoja. En la Figura 14 se ilustra este hecho, mostrando las trayectorias de los antepasados del vértice seleccionado hasta las hojas más alejadas de ellos. Por estas razones, el antepasado de un vértice seleccionado, en particular, debe lidiar con uno de estos dos escenarios: su nueva hoja más lejana proviene de una rama que no se vio afectada por la operación de eliminación o su nueva hoja más lejana proviene de la rama única que nace de la hija que conduce a la ruta que ha sido afectada por la operación de eliminación. De la misma manera, el valor *b* de esa hija se ve afectado por uno de estos dos escenarios.

La Figura 15 muestra los dos escenarios para el vértice n2, el cual tiene que decidir si su nueva hoja más lejana proviene de la rama de su hijo *n3* o de sus otras ramas. El problema es que tiene que esperar la computación del valor *b* de *n3* para realizar el suyo; por lo tanto, se puede ver la estructura

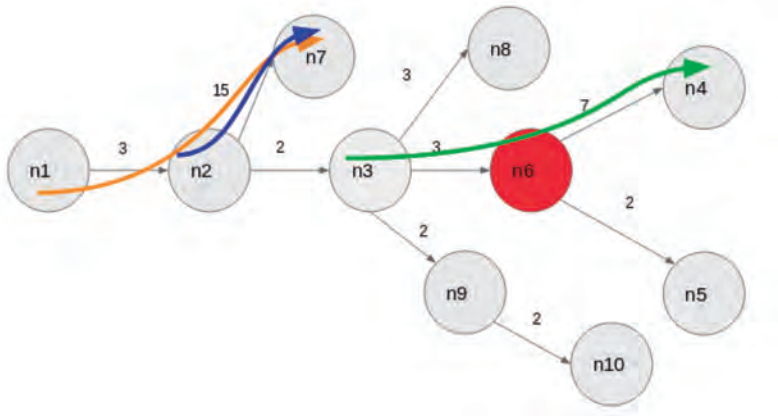


Figura 14. Actualización de los atributos b después de la eliminación del vértice 1

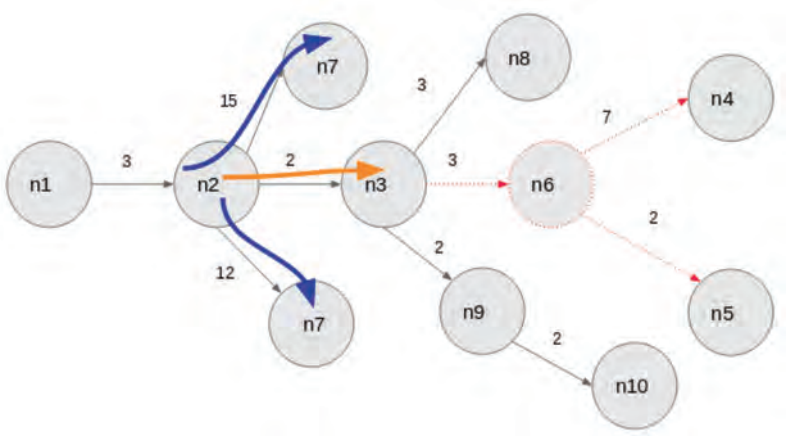


Figura 15. Escenarios para la nueva hoja más lejana

recursiva que debe ejecutarse secuencialmente para actualizar los valores de b, comenzando en el padre del vértice seleccionado, hasta el vértice raíz.

Usando el modelo computacional puro de Giraph, estas actualizaciones tomarían tantos superpasos como el número de ancestros de vértices seleccionados, sin embargo, utilizar la función master compute y las características de los aggregators es un enfoque diferente, donde a través de muchos superpasos, se recopila, en algún agregador, toda la información necesaria para hacer la actualización, de modo que en la función master compute sea posible calcular secuencialmente cada uno de los nuevos valores b, almacenarlos nuevamente en una variable global y usarlos para actualizar los valores b en el siguiente superpaso.

Para comprender todo este proceso, se deben considerar las dos opciones para actualizar el valor b del ancestro de un vértice seleccionado en particular, como es que la nueva salida más lejana puede provenir de las ramas que no se ven afectadas por la operación de eliminación o de la única rama que se ve afectada por ella. Desde el primer superpaso se empieza a trabajar en el primero de estos escenarios.

En las líneas 41 a 45 del Algoritmo 11, cada uno de los antepasados del vértice seleccionado envía su distancia a todos sus hijos. Ese valor, más el valor b actual del hijo, es igual a la distancia hasta la salida más lejana que sigue a esa rama. Luego, esta información debe devolverse al antepasado del vértice seleccionado, para que sea agregada con los resultados de las demás ramas.

En el superpaso 1, los vértices que recibieron mensajes pueden ahora completar la información de las distancias a las hojas más lejanas siguiendo sus respectivas ramas y enviar esa información a sus padres (Algoritmo 12, líneas 33-35). No obstante, los antepasados del vértice seleccionado que recibieron mensajes del superpaso anterior no deben informar la distancia a la hoja más lejana de la misma forma, ya que una de las ramas que nace de ellos ha cambiado y aún no conocen la ubicación de sus hojas más lejanas, por lo tanto, ninguno de los antepasados del vértice seleccionado debe enviar la información sobre el estado actual de sus hojas más lejanas. Sin embargo, estos vértices hacen el camino al vértice seleccionado desde el vértice raíz, y aunque estos vértices pueden saber que son parte de ese camino, no pueden saber cuál es el hijo que los lleva al vértice seleccionado. Es por ello que todos los antepasados de vértices seleccionados envían un mensaje con sus id a sus padres, para que cualquiera de ellos sepa cuál es la ruta para llegar al vértice seleccionado, cuyos vértices son los únicos que se verán afectados por la operación de eliminación (Algoritmo 12 líneas 12-15). Luego, para actualizar los valores b de estos vértices, se debe calcular el valor b para el padre del vértice seleccionado y este valor debe propagarse hacia arriba a través de la ruta de los ancestros del vértice seleccionado. En el superpaso 1, se inicia el trabajo de computación del nuevo valor b del padre del vértice seleccionado.

El computo del nuevo valor b del padre del vértice seleccionado –que es el valor clave para calcular todos los demás valores b de los antepasados del vértice seleccionado–, debe marcar las dos mismas opciones de todos los antepasados del otro vértice seleccionado, tiene que comprobar si la nueva salida más lejana proviene: de las ramas que no se ven afectadas por la operación de eliminación o de las nuevas ramas que se crearán conectando el padre del vértice seleccionado con los hijos del vértice seleccionado. La distancia a la hoja más lejana, siguiendo la última opción, se computa para ser usada en la operación de eliminación factible. Es importante recordar que este valor se almacena, en el caso 2 de la función master compute, en una variable global llamada parentBestPossibleNewB (Algoritmo 9, línea 24).

En el superpaso 2, la solución está lista para almacenar toda la información necesaria para producir los nuevos valores b de todos los antepasados de vértices seleccionados. Esta información se almacena en AllPredecessorsPossibleNewBsAggregator (Algoritmo 9, línea 25), cuyo valor es un mapa en el que las claves corresponden al id de cada uno de los ancestros

del vértice seleccionado y los valores a una estructura de datos denominada ElementsToComputeB, en donde: Int idParent es la identificación del padre del vértice; Dbl possNewB es el posible nuevo valor de b que vendría de la hoja más lejana de las ramas que no fueron afectadas por la operación de eliminación; y Dbl ptlPossNewBs es la distancia al único hijo que conduce al vértice seleccionado (este valor más el nuevo valor b de ese hijo es el otro posible valor b nuevo del vértice).

Para el ejemplo que se ilustró en la Figura 14, el valor de AllPredecessorsPossibleNewBsAggregator, cuando está completamente agregado, debe parecerse a la estructura que se presenta en el Código 6.

```
n1:     {
        idParent: nill
        possNewB: 0
        ptlPossNewBs: 3
},
n2:     {
        idParent: n1
        possNewB: 15
        ptlPossNewBs: 2
}
```

Código 6. AllPredecessorsPossibleNewBsAggregator

En el superpaso 2, mientras el padre del vértice seleccionado puede actualizar directamente su valor b, los antepasados del otro vértice seleccionado solo pueden almacenar información parcial en el aggregator ElementsToComputeB, la misma que solo se podrá completar en la siguiente invocación de la función master compute hasta que, secuencialmente, uno por uno, los nuevos valores b de los antepasados del vértice seleccionado se computen. En este superpaso, todos los antepasados del vértice seleccionados reciben los mensajes con los posibles nuevos valores b provenientes de sus ramas no afectadas, los que deben reducirse a un valor único, la distancia a la hoja más lejana que sigue esos caminos. En el caso del padre del vértice seleccionado, solo se tiene que comparar ese valor con el valor de parentBestPossibleNewB para actualizar su propio valor b (algoritmo 16, líneas 17-22); en cambio, los otros ancestros de vértice seleccionados solo pueden poner información en ElementsToComputeB

y deben almacenar el valor reducido de los mensajes entrantes de las ramas no afectadas en el campo possNewB. Por otro lado, con el mensaje entregado por

```
1: procedure createGlobalToUpdatePredeccesorsMasterCompute
2:     parentBestPossibleNewBAggregator.value ← max(parentPossibleNewBsAggregator.value)
3: end procedure
```

Algoritmo 15. Computación del valor b para el predecesor del vértice seleccionado (master compute 2)

```
1: procedure updateNodesAndBeginBestLocationComputation(vertex, msgs)
2:     if vertex.id == selectedNode.id then
3:         selectedNodeChildren ← []
4:         for edge in vertex.edges do
5:             vertex.removeEdge(edge)
6:             selectedNodeChildren.add(edge.target)
7:         end for
8:         removeEdgeRequest(vertex.predecessorId, vertex.id)
9:         selectedNodeChildrenAggregator.aggregate(selectedNodeChildren)
10:    else if vertex.isPredecessor(selectedNode) then
11:        maxB ← max(msgs.findAll("POSSB"))
12:        if vertex.id == selectedNode.predecessorId then
13:            vertex.removeEdge(selectedNode.id)
14:            for key in deleteCostForSuccessors.keySet do
15:                vertex.addEdge(key)
16:            end for
17:            if maxB > parentBestPossibleNewBAggregator.value then
18:                vertex.b ← maxB
19:            else
20:                vertex.b ← parentBestPossibleNewBAggregator.value
21:            end if
22:            parentBAggregator.aggregate((vertex.idParent, vertex.b))
23:        else
24:            possibleNewBUtilities.idParent ← vertex.idParent
25:            possibleNewBUtilities.possNewB ← maxB
26:            possibleNewBUtilities.partialPossNewB ← vertex.distances.get(msgs.get("ID"))
27:            AllPredecessorsPossibleNewBsAggregator.aggregate((vertex.id, possibleNewBUtilities))
28:        end if
29:    else if vertex.isSuccessor(selectedNode) then
30:        for key in deleteCostForSuccessors.keySet do
31:            if vertex.isSuccesor(key) or vertex.id == key then
32:                n5.f ← n5.f + deleteCostForSuccessors(key)
33:                if vertex.id == key then
34:                    vertex.parent ← selectedNode.parent
35:                end if
36:            end if
37:        end for
38:    end if
39:    if feasibleInsert(vertex, selectedNode) then
40:        vertex.partialBestCost ← vertex.distances.get(selectedNode.id)
41:    end if
42:    for edge in vertex.edges do
43:        vertex.sendMessage(edge.target, ('TO_SUCC', vertex.distances.get(edge.target)))
44:        vertex.sendMessage(edge.target, ('TO_SELEC', vertex.distances.get(selectedNode)))
45:    end for
46: end procedure
```

Algoritmo 16. Actualización de vértices e inicio de la operación de mejor ubicación (superpaso 2)

el hijo del antepasado del vértice seleccionado, se puede obtener el valor de ptlPossNewBs (Algoritmo 16, líneas 23-28).

Con la información de AllPredecessorsPossibleNewBsAggregator completa, es posible computar, en el caso 3 de la función master compute, los nuevos valores *b* de los ancestros del vértice seleccionado, lo que se logra como se muestra en el Algoritmo 17, al final de cuya ejecución, se habrá creado un nuevo aggregator llamado newBs, donde se almacena un mapa con el nuevo valor *b* para cada ancestro de vértice seleccionado, el cual se usa en el superpaso 3 para actualizar los valores *b* correspondientes (Algoritmo 18, líneas 2-6).

```
1:  RDCMSTMasterCompute.computeBValues = function()
2:      registerAggregator("bestLocationAggregator")
3:      registerAggregator("newBsAggregator")
4:      possibleNewBs ← AllPredecessorsPossibleNewBsAggregator.value
5:      parentId ← parentBAggregator.value.id
6:      bValue ← parentBAggregator.value.b
7:      while not possibleNewBs.isEmpty() do
8:          elementToComputeB ← possibleNewBs.get(parentId)
9:          possNewBFromSelectedNode ← elementToComputeB.partialPossibleB + bValue
10:         possNewBFromOtherBranches ← elementToComputeB.bFromOtherBranches
11:         if possNewBFromOtherBranches > possNewBFromSelectedNode then
12:             bValue ← possNewBFromOtherBranches
13:         else
14:             bValue ← possNewBFromSelectedNode
15:         end if
16:         newBsAggregator.value(parentId) ← bValue
17:         formerSuccessorId ← parentId
18:         parentId ← elementToComputeB.predecessorId
19:         possibleNewBs.remove(formerSuccessorId)
20:     end while
21: end function
```

Algoritmo 17. Computación de los valores b para los predecesores del vértice seleccionado (master compute 3)

```
1:  procedure finishBestLocationComputation(vertex, msgs)
2:      if vertex.positions.get(selectedNode.id) = PREDECESSOR then
3:          if not vertex == selectedNode.parent then
4:              vertex.b = newBsAggregator.get(vertex.id).value
5:          end if
6:      end if
7:      parentToSelectedNode ← msgs.get("TO_SELEC")
8:      parentToHere ← msgs("TO_SUCC")
9:      cost ← parentToSelectedNode + selectedNode.distances.get(vertex.id) − parentToHere
10:     if feasibleInsert(vertex, selectedNode, cost) then
11:         partialBestLocation = Location(vertex, "FROM_NODE", vertex.partialBestCost)
12:         if cost < vertex.partialBestCost then
13:             partialBestLocation = Location(vertex, "BREAKING_EDGE", cost)
14:         end if
15:         bestLocationAggregator.aggregate(partialBestLocation)
16:     end if
17: end procedure
```

Algoritmo 18. Finalización de la operación de mejor ubicación (superpaso 3)

Operación de inserción

Esta operación se compone de dos suboperaciones: la de mejor ubicación, que se encarga de calcular el lugar donde se va a insertar el vértice seleccionado; y la de actualización del estado del grafo, producida por la propia operación de inserción.

Suboperación de mejor ubicación

Para cada vértice hay dos formas posibles de insertar el vértice seleccionado: directamente, como un hijo hoja del vértice (ATLEAF); o como padre del vértice, rompiendo la arista existente entre el vértice y su padre anterior (ATEDGE). La Figura 16 muestra ambos tipos de ubicación para el vértice n3.

En pocas palabras, la estrategia seleccionada para encontrar la mejor ubicación es así: para cada vértice del grafo, se computa el costo de insertar el vértice seleccionado en ambas ubicaciones; luego, nuevamente para cada vértice, se elige la mejor ubicación "local", la de menor costo, entre ATLEAF y ATEDGE; y finalmente, usando un agregador, se elige la mejor ubicación entre todos los vértices del grafo.

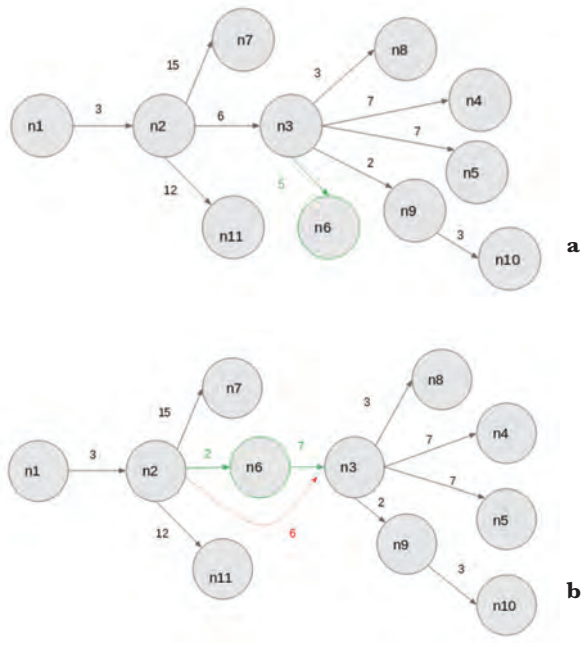


Figura 16. Localización para el vértice n3: ATLEAF (a); y ATEDGE (b)

En Giraph, esta suboperación se realiza así: en el superpaso 2 se computa el costo de insertar el vértice seleccionado ATLEAF en cada vértice –este costo solo se computa si la inserción en esa ubicación en particular es factible, es decir si el valor *f* del vértice más la distancia al vértice seleccionado es menor o igual que la restricción *lambda*–, y se almacena parcialmente en la variable de un vértice (Algoritmo 16, líneas 39-41). En el superpaso 2 no es posible computar el costo de insertar el vértice seleccionado ATEDGE, ya que el vértice no tiene la información de las aristas salientes del padre del vértice que debe insertarse o eliminarse, por lo tanto, en este superpaso, cada vértice debe enviar a sus hijos tanto la distancia entre él y ellos como la distancia entre él y el vértice seleccionado (Algoritmo 16, líneas 42-45).

En master compute 3, para agregar la mejor ubicación "local" que computa cada vértice, se crea BestLocationAggregator. Una ubicación está definida por un vértice y una forma de inserción y tiene como atributos: int vId, el ID del vértice; enum way, la forma de inserción (ATLEAF o ATEDGE); y double cost, el costo de insertar el vértice seleccionado en esa ubicación.

En el superpaso 3 se computa el costo de insertar el vértice seleccionado ATEDGE para cada vértice, sumando y restando el peso de las aristas que mutarían debido a la inserción (Algoritmo 18, líneas 7-9); una vez calculado ese costo, es posible verificar la factibilidad de la operación sumando el valor *f* del vértice, el valor *b* del vértice y el costo de la inserción, y verificando que esa suma sea menor o igual que la restricción *lambda* (Algoritmo 18, línea 10); luego, se decide la mejor forma de inserción (ATLEAF o ATEDGE) y se agrega esta mejor ubicación local para obtener la mejor ubicación global en el siguiente superpaso (Algoritmo 18, líneas 11 a 15).

```
1: procedure BESTLOCATIONAGGREGATION(newValue)
2:     if newValue.cost < bestLocationAggregator.value.cost then
3:         bestLocationAggregator.value ← newValue
4:     end if
5: end procedure
```

Algoritmo 19. Agregado en la suboperación de mejor ubicación

Consecuencias de la operación de inserción en el estado del grafo

Los cambios en el estado del grafo después de la operación de inserción depende de tres escenarios: si el costo total del movimiento es mayor que 0, significa que la solución anterior era mejor y por lo tanto se debe restaurar el estado

del grafo al momento anterior al movimiento; en los otros dos escenarios se realiza la operación de inserción, sin embargo, tanto la actualización de *b* y *f*, como el vector de posiciones y la topología del grafo varían dependiendo de si la inserción es ATLEAF o ATEDGE. Para mayor claridad, el superpaso 4 se ha dividido en tres algoritmos, uno para cada escenario: restaurar (Algoritmo 20), inserción con ATLEAF (Algoritmo 21) e inserción con ATEDGE (Algoritmo 22).

```
1: procedure RESTOREPREVIOUSMOVEMENT(vertex, msg)
2:     vertex.b ← vertex.oldB
3:     vertex.f ← vertex.oldF
4:     if vertex == selectedNode then
5:         for child in selectedNodeChildrenAggregator.value do
6:             vertex.addEdge(child)
7:         end for
8:     else if vertex == selectedVertex.parent then
9:         vertex.addEdge(selectedNode)
10:        for child in selectedNodeChildrenAggregator.value do
11:            vertex.removeEdge(child)
12:        end for
13:    end if
14: end procedure
```

Algoritmo 20. Restaurar (superpaso 4)

```
1: procedure INSERTOPERATIONANDUPDATENODESCOMPUTATION(vertex, msg)
2:     if vertex == selectedNode then
3:         if bestLocation.way == "FROM_NODE" then
4:             vertex.parent ← bestLocation.node
5:         end if
6:     end if
7:     if vertex.isPredecessor(bestLocation.node) then
8:         vertex.positions.get(selectedNode.id) = PREDECESSOR
9:         sendMessage(vertex.parent, ("ID", vertex.id))
10:        sendMessage(selectedNode, (vertex.id, PREDECESSOR))
11:    else if vertex.isSuccessor(bestLocation.node) then
12:        if bestLocation.way == "FROM_NODE" then
13:            vertex.positions(selectedNode) = NONE
14:            sendMessage(selectedNode, (vertex.id, NONE))
15:        end if
16:    else if vertex == bestLocation.node then
17:        if bestLocation.way == "FROM_NODE" then
18:            if bestLocation.cost > vertex.b then
19:                vertex.b ← bestLocation.cost
20:            end if
21:            vertex.positions.get(selectedNode.id) = PREDECESSOR
22:            vertex.addEdge(selectedNode.id)
23:            selectedNodeF ← vertex.f + vertex.distances.get(selectedNode.id)
24:            sendMessage(selectedNode, ("F", selectedNodeF))
25:            sendMessage(vertex.parent, ("ID", vertex.id))
26:        end if
27:    else
28:        vertex.positions.get(selectedNode) = NONE
29:        sendMessage(vertex.parent, ("PAR_B", vertex.b))
30:    end if
31: end procedure
```

Algoritmo 21. Actualización y operación de inserción ATLEAF (superpaso 4)

```
1: procedure INSERTOPERATIONANDUPDATENODESCOMPUTATION(vertex, msg)
2:     if vertex == selectedNode then
3:         if bestLocation.way == "BREAKING_EDGE" then
4:             vertex.parent = bestLocation.node.parent
5:             vertex.addEdge(bestLocation.node)
6:             parentBestPossibleNewBAggregator.aggregate(vertex.distances.get(bestLocation.id))
7:         end if
8:     end if
9:     if vertex.isPredecessor(bestLocation.node) then
10:        vertex.positions.get(selectedNode.id) = PREDECESSOR
11:        sendMessage(vertex.parent, ("ID", vertex.id))
12:        sendMessage(selectedNode, (vertex.id, PREDECESSOR))
13:        if vertex == bestLocation.node.parent then
14:            if bestLocation.way == "BREAKING_EDGE" then
15:                vertex.addEdge(selectedNode.id)
16:                vertex.removeEdge(bestLocation.node.id)
17:                parentBestPossibleNewBAggregator.aggregate(vertex.distances.get(selectedNode.id))
18:                selectedNodeF ← vertex.f + vertex.distances.get(selectedNode.id)
19:                sendMessage(selectedNode, ("F", selectedNodeF))
20:            end if
21:        end if
22:    else if vertex.isSuccessor(bestLocation.node) then
23:        if bestLocation.way == "BREAKING_EDGE" then
24:            vertex.f ← vertex.f + bestLocation.cost
25:            vertex.positions.get(selectedNode) = SUCCESSOR
26:        end if
27:    else if vertex == bestLocation.node then
28:        if bestLocation.way == "BREAKING_EDGE" then
29:            vertex.f ← vertex.f + bestLocation.cost
30:            vertex.positions.get(selectedNode) = SUCCESSOR
31:            vertex.parent = selectedNode
32:            parentBestPossibleNewBAggregator.aggregate(vertex.b)
33:            sendMessage(selectedNode, ("BEST_LOCATION_B", vertex.b))
34:        end if
35:    else
36:        vertex.positions.get(selectedNode) = NONE
37:        sendMessage(vertex.parent, ("PAR_B", vertex.b))
38:    end if
39: end procedure
```

Algoritmo 22. Actualización y operación de inserción ATEDGE (superpaso 4)

En el primer escenario, cuando el mejor movimiento no mejora la solución actual, se mantiene el costo de un movimiento completo en el MovementCostAggregator. Antes de realizar la suboperación de mejor ubicación, solo se ha computado el costo de la operación de eliminación, no obstante, debido a que la mejor operación de ubicación calcula el costo de la operación de inserción, el costo total del movimiento se puede conocer simplemente agregando el mejor costo de ubicación al valor del MovementCostAggregator en el master compute 4 (Algoritmo 9, línea 34).

Si este costo agregado es mayor que 0, significa que el movimiento del vértice seleccionado no puede mejorar la solución actual y, por lo tanto, se debe restaurar el estado del grafo; en este escenario, se emite una variable booleana para notificar a todos los vértices que el movimiento se abortó. Los

únicos cambios que se deben revertir corresponden a las consecuencias de la operación de eliminación, por lo que se deben restaurar los valores b y f de todos los vértices y el estado topológico del grafo, insertando el vértice seleccionado donde estaba antes de la operación de eliminación.

Para restaurar los valores de f y b, se hace una copia de ellos antes de la operación de borrado (Algoritmo 12, líneas 8-9), porque en caso de abortar el movimiento, hay que restaurarlos (Algoritmo 20, líneas 2-3). Por otra parte, para restaurar el estado topológico y gracias al selectedNodeChildrenAggregator, el vértice seleccionado agrega una arista a cada uno de sus hijos anteriores, y el padre del vértice seleccionado elimina las aristas de los mismos vértices y agrega una arista al vértice seleccionado (Algoritmo 20, líneas 4-13). Todo el código anterior se ejecuta en el superpaso 4, por ello, en el siguiente superpaso se puede iniciar normalmente un nuevo movimiento.

El segundo escenario, la inserción ATLEAF, es la forma más sencilla de inserción, por ello, implica menos cambios en el estado del grafo. En primer lugar, el cambio topológico se realiza simplemente insertando una arista entre el vértice de mejor ubicación y el vértice seleccionado (Algoritmo 21, línea 22), no hay que actualizar el valor f de ningún vértice en el grafo, excepto en el vértice seleccionado en sí, porque no se afecta ninguna de las rutas desde los vértices hasta la raíz. Por lo tanto, el mejor vértice de ubicación solo tiene que enviar su propio valor f más la distancia al vértice seleccionado (Algoritmo 21, líneas 23-24), para que el vértice seleccionado pueda reemplazar su valor f con el valor del mensaje en el siguiente superpaso (Algoritmo 11, línea 9). Además, aunque la actualización de los valores b es muy similar a la operación de eliminación, ninguna de las rutas actuales a las hojas se ve afectada por la operación, por lo que cada antepasado solo tiene que verificar si la distancia al vértice seleccionado es mayor que su valor b real. Sin embargo, aunque este parece ser un procedimiento más sencillo, tiene que utilizar prácticamente el mismo algoritmo que se presentó en la sección de consecuencias de la operación de eliminación.

El superpaso 4 comienza entonces actualizando el valor b del padre del vértice seleccionado, al mismo tiempo, los ancestros del vértice de mejor ubicación envían sus identificadores a sus padres para desplegar el camino hacia la nueva hoja (Algoritmo 21, línea 25). Desde el siguiente superpaso (superpaso 0), la actualización se realiza de la misma forma que la operación de eliminación, con la diferencia de que para cada antepasado, su correspondiente

elementsToComputeB.possNwBs es igual a su valor *b* actual, porque ninguno de los caminos a las hojas se ve afectado por la operación; por lo tanto: en el master compute 1 (Algoritmo 9, línea 12) se computan los nuevos valores *b* y por el superpaso 1, todos los antepasados han actualizado su valor *b* (Algoritmo 12, líneas 4-6).

Finalmente, se debe lidiar con una variable que no se ha modificado hasta ahora, después de la operación de inserción las posiciones de todos los vértices sobre el vértice seleccionado han cambiado, así como las posiciones del vértice seleccionado sobre todos los demás vértices, por lo tanto, todos los nuevos ancestros del vértice seleccionado (el mejor vértice de ubicación y todos sus ancestros) tienen que actualizar sus posiciones sobre el vértice seleccionado con la etiqueta ANCESTOR (Algoritmo 21, líneas 8 y 21), y todos los demás vértices en el grafo deben hacer lo mismo, pero con la etiqueta NONE (Algoritmo 21, líneas 13 y 28).

Por otro lado, todos los vértices del grafo envían un mensaje al vértice seleccionado con la posición que necesita actualizar en su propia matriz de posiciones en el siguiente superpaso. Los antepasados del vértice seleccionado envían la etiqueta DESCENDANT y los vértices que no son antepasados ni descendientes envían la etiqueta NONE. Cada uno de estos mensajes es enviado en una tupla con el id del vértice emisor y la etiqueta, la cual se usa en el superpaso 0 para actualizar la matriz de posiciones del vértice seleccionado (Algoritmo 11, líneas 4-8).

La del tercer escenario, la inserción ATEDGE, es ligeramente más compleja porque afecta a más vértices en el grafo, los cambios topológicos se realizan tanto por el vértice seleccionado como por el padre del vértice de mejor ubicación (Algoritmo 22, líneas 5, 15 y 16).

Todos los descendientes del nuevo vértice seleccionado tienen que actualizar su valor *f* agregando el mejor costo de ubicación a sus valores *f* actuales (Algoritmo 22, líneas 24 y 29). Estos vértices son el mejor vértice de ubicación y todos sus descendientes. Adicionalmente, el valor *f* del vértice seleccionado debe actualizarse, tal como se hizo en la inserción ATLEAF. La inserción ATEDGE afecta la distancia desde el antepasado del nuevo vértice seleccionado hasta algunas de las hojas, por lo tanto, se puede utilizar el concepto de ramas afectadas y no afectadas para actualizar sus valores *b*. La idea es que el ancestro de cada nuevo vértice seleccionado verifique si la nueva hoja más lejana proviene de las ramas no afectadas o de la rama afectada.

Para calcular la hoja más lejana procedente de las ramas no afectadas, los vértices que no son ancestros ni descendientes del vértice de mejor ubicación envían sus valores b a sus padres (Algoritmo 22, línea 37). En el siguiente superpaso, los nuevos descendientes del vértice seleccionado pueden computar la distancia a las hojas más lejanas siguiendo sus ramas no afectadas, adicionando la distancia al vértice emisor al valor del mensaje en sí. Luego, se elige el mayor de estos valores y se almacena en elementsToComputeB.possNwBs (Algoritmo 11, línea 28).

Por otro lado, a diferencia de la operación de eliminación, solo hay una rama del padre del nuevo vértice seleccionado que se ve afectada por la operación de inserción; en consecuencia, solo se debe comparar la distancia a la hoja más lejana que sigue esta rama con la calculada a partir de las ramas no afectadas, para lo que se usa parentBestPossibleNewB, cuya función aggregate simplemente suma los valores dobles. Por lo tanto, se tienen que sumar el valor b del mejor vértice de ubicación y la distancia desde el padre del nuevo vértice seleccionado hasta el vértice de mejor ubicación que cruza las dos nuevas aristas (Algoritmo 22, líneas 6, 17 y 32). Una vez que se ha actualizado el nuevo valor b del padre del vértice seleccionado (Algoritmo 11, líneas 18-24), el procedimiento para actualizar los valores b de todos los antepasados del vértice seleccionado es el mismo que en las operaciones de eliminación e inserción ATLEAF. Finalmente, para actualizar el valor b del vértice seleccionado, en el superpaso 4, el vértice de la mejor ubicación envía su propio valor b (Algoritmo 22, línea 33), el cual tiene que ser usado en el siguiente superpaso por el vértice seleccionado para realizar la actualización (Algoritmo 11, línea 18).

La actualización de la matriz de posiciones es muy similar a la inserción ATLEAF, excepto que los descendientes del nuevo vértice seleccionado tienen también que actualizar sus posiciones sobre el vértice seleccionado y enviarle un mensaje para permitir la actualización de su propia matriz.

EVALUACIÓN

Los experimentos se realizaron utilizando un clúster Hadoop con quince nodos de procesamiento, cada uno equipado con un procesador Intel Core I7-3770 CPU @ 3.40GHz con 8 núcleos y 16 GB de memoria RAM. Giraph se configuró con catorce trabajadores y un nodo maestro. Además, se utilizaron dos servidores para ejecutar los servicios maestros HDFS y YARN.

La Figura 17 muestra una abstracción del diagrama de implementación que se centra en los servicios de Giraph. Se implementaron catorce nodos de procesamiento –de hgrid1 a hgrid14–, con componentes de trabajo y se alojó el componente maestro en uno –el hgrid15–. En el diagrama se incluyen los servidores HDFS y YARN, grid100 y grid101.

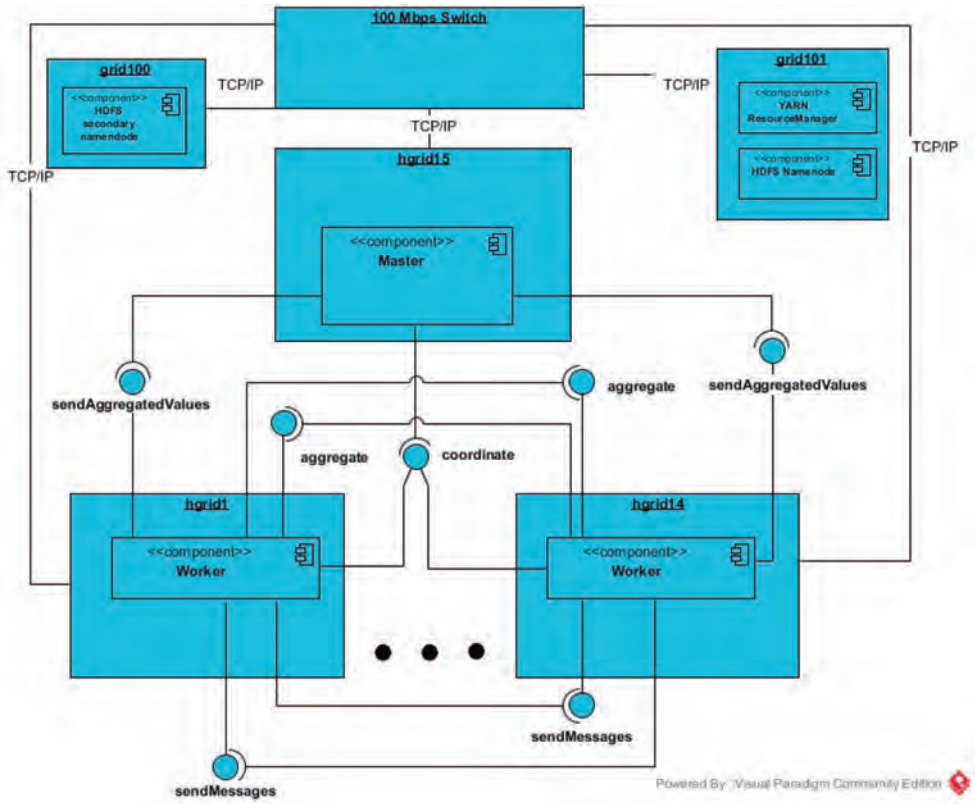


Figura 17. Diagrama de despliegue

La mayoría de los datos utilizados en este proyecto se basan en abstracciones de las capas de la red LR-PON de fibra óptica de España. La red conecta los sitios de intercambio con los clientes finales. En las abstracciones, los vértices representan subredes, que son tanto vértices como aristas disjuntas entre sí. La red de abstracción tiene 18.819 vértices, que incluyen los sitios de intercambio. Uno de los sitios de intercambio se seleccionó como vértice raíz. Para cada vértice se tienen las coordenadas GPS desde donde se puede construir un

grafo completo calculando distancias euclidianas entre sí. Las coordenadas se extienden sobre un área cercana a la totalidad del área continental de España (alrededor de 500.000 km2).

Como solución inicial, se construyó un árbol en el que todos los vértices están conectados directamente a la raíz. En distancias euclidianas, esta sencilla solución garantiza que se cumplan todas las restricciones, dado que la distancia desde la raíz a todos los vértices se minimiza. Durante los experimentos, sin embargo, se exploraron otras alternativas.

En cuanto a la restricción de distancia (λ), se tuvo en consideración que en una instancia euclidiana el problema no es viable si *lambda* está restringida a ser menor que la distancia euclidiana entre la raíz y su vértice más lejano, por lo que este número es su límite inferior; asimismo, si *lambda* es mayor que el diámetro del árbol de recubrimiento mínimo (MST, *Minimum Spanning Tree*) del grafo, la solución podría encontrarse en tiempo polinomial, con lo que utilizar el enfoque propuesto no tendría sentido, entonces, el diámetro del MST es el límite superior de *lambda*. *Lambda* fue definida como el punto medio de sus límites inferior y superior. En la presentación de esta sección, se usa esta definición de *lambda*, a menos se indique explícitamente lo contrario.

FASE DE EXPERIMENTOS PILOTO

En Giraph es posible establecer la cantidad de subprocesos que usa un trabajador: la carga de trabajo de entrada asignada a cada uno se divide entre los subprocesos y cada uno calcula su grupo de vértices al mismo tiempo. En esta fase, se trató de encontrar el número óptimo de subprocesos por trabajador de manera experimental, para ello, partiendo del conjunto de datos original, se realizaron diez mil movimientos a partir de la solución inicial sencilla, con un tamaño de grafo de entrada dado, y se midieron los tiempos. Los resultados de estos experimentos se presentan en la Figura 18; aunque no se obtuvieron diferencias significativas para la configuración óptima, se decidió utilizar las dos mejores configuraciones –cuatro y doce subprocesos por trabajador– en los experimentos de las fases posteriores.

Durante la fase piloto también se verificó cuánto se podía hacer crecer el conjunto de datos para realizar experimentos razonables en términos de tiempo: primero, con base en los resultados de los artículos publicados, se acordó que realizar al menos tantos movimientos como vértices tiene un grafo,

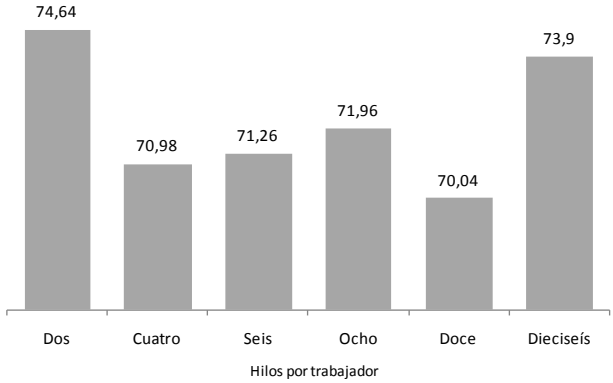


Figura 18. Cantidad de hilos por trabajador realizando diez mil movimientos (minutos)

sería suficiente para notar mejoras respecto de la solución inicial y para verificar el rendimiento del algoritmo. Se tomaron entonces, la mitad, la totalidad y una y media veces el conjunto de datos de España, su procesamiento demoró duró 32 minutos, algo más de dos horas y casi cinco horas, respectivamente.

El *dataset* con la mitad de los datos se formó con los primeros 9.409 puntos del *dataset* original [23]; para el de 1.5 veces, se tomó la mitad de los puntos y se trasladó a la derecha, en el eje *x*, una distancia equivalente a la que hay entre los puntos más a la izquierda y más a la derecha del conjunto de datos, y se agregó al *dataset* original. Dado lo extenso del tiempo que tomo el procesamiento de 1.5 veces los datos originales, se decidió hacer los experimentos solo con estos tres conjuntos de datos, los mismos que se etiquetaron como: media España, España y España y media, respectivamente. Cabe señalar que, aun cuando se aumento el número de vértices en un factor de 0.5, el número de aristas se hizo en un factor de 2.25, toda vez que se están considerando grafos completos. En la Tabla 1 se presenta la composición y tamaño de los *datasets* derivados de la red España.

Tabla 1. Tamaño de los conjuntos de datos

Conjunto de datos	Vértices	Aristas	Tamaño de archivo
España	18,819	354,135,942	1.2 GB
Media España	9,409	88,519,872	4.6 GB
España y media	28,228	796,791,756	11 GB

INSTANCIAS PEQUEÑAS

Se realizaron algunos experimentos en instancias pequeñas para verificar que la solución fuera correcta, para ello se generaron dos grafos aleatorios completos con 500 y 1.000 vértices, se ejecutó el algoritmo durante 30 minutos y se seleccionó el mejor mínimo local encontrado durante ese tiempo; también se usó la formulación de programación entera mixta del problema que presentaron Arbeláez et al. [1] para ejecutar el solucionador de CPLEX –una implementación patentada del método Simplex en el lenguaje de programación C–, en las mismas instancias. CPLEX se ejecutó usando una licencia institucional en un servidor equipado con un procesador Intel Xeon CPU E5-2620 v4 @ 2.10GHz con 32 núcleos y 60 GB de memoria RAM. En la Tabla 2 se presentan los resultados de las instancias pequeñas que utilizan los enfoques Giraph y CPLEX.

Tabla 2. Ejecución durante 30 minutos en pequeñas instancias aleatorias

Enfoque	Vértices	Costo inicial	Costo final	Mejora relativa	Lambda	*b* de raíz
ILC distribuido	500	253,72	13,81	94,56 %	0,86	0,33
	1000	500,19	28,99	94,20 %	0,99	0,66
CPLEX	500	253,72	1,03	99,59 %		
	1000	500,19	1,05	99,79 %		

En todos los experimentos se verificó la corrección de la solución, la cual fue evidente porque el valor *b* de la raíz –la distancia desde ella hasta su hoja más lejana–, fue menor que la restricción *lambda*. Además, se definió:

- el costo final, como el costo de la solución subóptima después de la ejecución del algoritmo;
- el costo inicial como el costo de la solución inicial; y
- la mejora relativa, como la diferencia entre ambos dividida entre el costo inicial: *(Costo inicial – Costo final) / Costo inicial*.

Como se esperaba, el enfoque CPLEX superó significativamente la implementación desarrollada en el proyecto, lo que ofrece una primera pista de cuán grande debe ser una instancia para aprovechar el enfoque distribuido propuesto.

INSTANCIAS GRANDES

Estos son los experimentos que realmente evalúan la implementación desarrollada. Se decidió experimentar en tres dimensiones, modificando: el tamaño de la instancia, para lo que había tres valores disponibles: el número de subprocesos por trabajador, con dos posibilidades; y el valor de la restricción *lambda*, para la que se usaron tres valores (media, izquierda y derecha). La *lambda* media es la misma definida al inicio de esta sección de evaluación; la *lambda* izquierda es el punto medio entre el límite inferior de *lambda* y la *lambda* media; y la *lambda* derecha es el punto medio entre la *lambda* media y la *lambda* superior. Los tiempos de ejecución de los experimentos se resumen en la Tabla 3 y el desempeño, expresado en términos de costo, en la Tabla 4, en ellas se evidencia cómo la implementación desarrollada logró resultados útiles en

Tabla 3. Tiempo de ejecución de los experimentos (minutos)

Hilos por trabajador	Conjunto de datos	Lambda izquierda	Lambda media	Lambda derecha
4	España	130.38	135.20	131.00
	Media España	33.35	32.64	32.71
	España y media	284.20	277.51	280.00
12	España	127.47	127.63	131.02
	Media España	32.91	33.56	32,87
	España y media	277.98	276.33	278.12

Tabla 4. Resultados de los experimentos en términos de costo

Restricción de distancia	Instancia	Costo inicial	Costo final	Mejora relativa
Lambda izquierda	España	72.564,46	34.944,42	51,84 %
	Media España	36.219,45	17.405,82	51,94 %
	España y media	202.952,00	100.484,85	50,49 %
Lambda media	España	72.564,46	35.868,64	50,57 %
	Media España	36.219,45	17.981,83	50,35 %
	España y media	202.952,00	100.434,38	49,49 %
Lambda derecha	España	72.564,46	36.021,84	50,36 %
	Media España	36.219,45	17.962,31	50,41 %
	España y media	202.952,00	99.955,80	50,75 %

un tiempo de proceso razonable, con mejoras relativas superiores al 50 % en instancias con un tamaño de archivo superior a 11 GB.

Además, aunque el propósito de esta evaluación no era comprobar el rendimiento de la búsqueda local por iteración, se demostró –y se reafirmó con los siguientes experimentos– que solo necesita un pequeño número de movimientos para realizar cambios notables en la solución subóptima inicial. A partir de estos resultados, se sabe además que el tiempo necesario para alcanzar los mínimos locales en estos casos se debe medir en términos de días.

Los datos recopilados también muestran la irrelevancia de la restricción de Lamba durante los primeros movimientos. Se esperaba que las instancias más restringidas por la *lambda* izquierda tuvieran un costo final mayor que las instancias más relajadas, porque la mayor amplitud del espacio de búsqueda del segundo, aumenta las posibilidades de encontrar mejores soluciones, sin embargo, no se observó este patrón, probablemente porque se realizaron muy pocas iteraciones, menos de las necesarias para ver un comportamiento que se espera en una etapa convergente del algoritmo.

Se quería tener además alguna medida de rendimiento, para lo que se tomó en cuenta que un movimiento tiene una complejidad $O(n)$, que siempre está determinada por la mejor operación de ubicación –como se explicó en la presentación del modelo computacional Giraph–. Por ello, se puede computar la cantidad total de trabajo multiplicando los movimientos realizados por el número de vértices (n2); al dividir este valor entre el tiempo de ejecución del experimento se obtiene la medida buscada. En otras palabras, se computa aproximadamente el número de ubicaciones evaluadas por minuto. En la Figura 19 se muestra la medida de rendimiento para los experimentos con la *lambda* media y doce subprocesos por trabajador. Estos resultados revelan la naturaleza del *framework* Giraph, pues a medida que una instancia se hace más grande, el algoritmo funciona mejor, por abordar tareas que implican un uso más intensivo de CPU. Esto sucede porque el número de trabajadores es fijo, de esa manera, si hay más vértices, cada trabajador tiene que ejecutar más funciones de computo. Sin embargo, se puede observar que probablemente las instancias usadas son aún muy pequeñas para aprovechar la implementación desarrollada (aunque eran demasiado grandes dada la restricción de tiempo de este proyecto).

Como se explicó en la presentación del modelo, en algunas fases del algoritmo los vértices tienen que iterar sobre sus hijos, por lo tanto, al principio, la solución

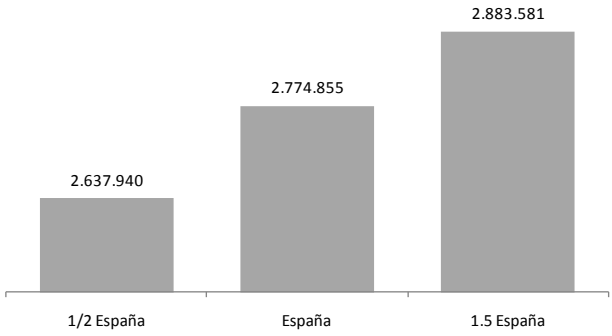


Figura 19. Ubicaciones evaluadas por minuto

inicial sencilla no va a explotar el paralelismo correctamente, porque con un solo padre, el vértice raíz tiene que hacer todo el trabajo, mientras que los otros vértices simplemente esperan. En cambio, una estructura de grafo más alargada, con los hijos distribuidos entre varios padres, va a ser más eficaz.

En consecuencia, se construyeron dos tipos adicionales de soluciones iniciales: en el enfoque más simple, se ejecutó MST sobre el grafo, se eliminaron los vértices que violan la restricción y se reconectaron directamente al vértice raíz; en el otro enfoque, se dividió aleatoriamente el grafo en particiones de un tamaño que el algoritmo BKRUS pueda manejar y se ejecutó dicho algoritmo en cada partición, incluyendo siempre el vértice fuente. Finalmente, se fusionaron los resultados. El Algoritmo 23 ilustra esta estrategia.

```
1: T ← empty graph
2: while G is not empty do
3:     N ← pick k nodes from G (different from source)
4:     G' ← projection of G on N ∪ source
5:     T'← BKRUS(G', source, λ)
6:     T ← T ∪ T'
7:     remove N from G (and its corresponding edges)
8: end while
```

Algoritmo 23. Solución inicial BKRUS (G, source, λ)

Ambas estrategias fueron computadas en procesos en serie en una sola máquina. Por una parte, hubo problemas al ejecutar el MST en la instancia más grande, pues se requería más de los 64GB de memoria física disponible y

por ello se tuvo que usar espacio de intercambio, lo que hizo que la ejecución demorara más de una hora; por otra, para los casos más importantes, la ejecución de la estrategia BKRUS tomó varias decenas de minutos.

Los experimentos se repitieron usando estas soluciones iniciales, pero variando el tamaño de la instancia, usando doce subprocesos por trabajador y la *lambda* media. Los resultados (Tabla 5) confirman lo esperado: tanto en BKRUS como en el MST reparado, el tiempo de ejecución se redujo entre 8 % y 12 %, a pesar de que hubo un número considerable de vértices reparados en el enfoque MST reparado (lo que contribuye al problema que se trata de resolver).

Tabla 5. Resultados usando varias soluciones iniciales

Estrategia	Instancia	Vértices reparados	Tiempo (minutos)	Costo inicial	Costo final	Mejora relativa
MST reparados	España	2,345	117,98	14.723,99	8.426,41	42,77 %
	Media España	3,640	29,73	19.129,09	10.390,97	45,68 %
	España y media	4,849	246,78	76.989,65	43.106,15	44,01 %
BKRUS	España		119,20	2.758,16	2.230,92	19,12 %
	Media España		32,93	1.168,88	988,97	15,39 %
	España y media		252,70	3.830,30	3.181,96	16,93 %

Lo anterior permite pensar que en instancias aún más alargadas, el algoritmo puede ser más eficaz. Además, los resultados de la Tabla 5 relacionados con el costo demuestran que el enfoque puede usarse como una herramienta complementaria para mejorar soluciones que ya sean buenas. El costo de la solución inicial de MST reparado lo mejoró en alrededor de 50 % y el de las soluciones iniciales provenientes de BKRUS –un algoritmo especializado para BKRUS–, en un 20 %. Este último porcentaje podría parecer pequeño, pero un 20 % de mejora puede representar una cantidad significativa de recursos durante la implementación de una red de comunicación óptica, por ejemplo.

ANÁLISIS DE RESULTADOS

Uso de Apache Giraph

Una de las ideas preliminares en este proyecto fue usar más de un *framework* para distribuir únicamente la exploración del vecindario y mantener las operaciones

restantes en un proceso en serie. El uso de MapReduce o *framework*s similares para lograr dicha distribución fue la principal opción considerada al inicio del proyecto. Las cosas no cambiaron mucho al estudiar GraphX, otro *framework* de procesamiento de grafos distribuido, que por sus limitaciones de expresividad se decidió mantener para usarlo como herramienta complementaria, solo para una parte específica de la estrategia. A modo de ejemplo, GraphX no permite la mutación de grafos; pero Giraph, primordialmente gracias a sus funciones de Master compute y Shared State, permite implementar toda la búsqueda local iterada utilizando un solo *framework*, y con ello evita gastos generales (*overhead*) potenciales.

La actualización del estado después de una mutación de grafo, que fue una de las operaciones más laboriosas de lograr, sería demasiado engorrosa sin un entorno de ejecución de procesos en serie y una forma de mantener un estado global en los nodos de procesamiento; por lo tanto, la expresividad en Giraph sirve adecuadamente para implementar algoritmos de grafos en los que se muta un grafo y se quiere monitorear la posición de cada vértice o de cualquier variable relacionada con él. La solución desarrollada ofrece una estrategia detallada para lograr esto, sin embargo, la API de Giraph aún se puede mejorar para resolver este tipo de problemas.

La única manera de implementar shared state en Giraph es mediante la filosofía del agregador, según ella, se debe utilizar una función conmutativa y una función asociativa para almacenar datos. Sin embargo, no siempre es deseable agregar variables globales propias; por ejemplo, con el agregador parentPossibleNewBsAggregator (Algoritmo 14), se quiere almacenar un mapa *hash* que permita su extensión y la actualización de sus elementos; sin embargo, el almacenamiento de una estructura de este tipo utilizando un agregador conlleva penalizaciones de rendimiento, porque en cada invocación de un función Aggregate, se debe atravesar toda la estructura –y se invoca varias veces–. Ampliar Giraph con una estructura de datos distribuida, especializada, sin las limitaciones de los agregadores, ayudaría a construir una mejor implementación de la estrategia propuesta por el proyecto.

Para finalizar el análisis de Giraph, cabe resaltar la granularidad flexible que se puede lograr con este *framework*, gracias al paradigma de programación centrada en vértices, según el cual, el tamaño de grano está determinado por el número de vértices que tiene asignado un trabajador. Para ser precisos, la cantidad total de trabajo de cada trabajador antes de la comunicación es la

ejecución de la función compute de todos los vértices; entonces, gracias a la partición del grafo realizada por Giraph, se puede afirmar que la solución desarrollada en el proyecto permite una distribución automáticamente uniforme y balanceada, sin importar la cantidad de nodos de procesamiento.

Una implementación para Big Data

La estrategia distribuida desarrollada se evaluó en diferentes escenarios, los que incluyeron instancias aleatorias no euclidianas, una instancia de red LP-RON real y variaciones de ella, para establecer los límites de este proyecto. La implementación permite trabajar con instancias que superan la capacidad de memoria de una sola máquina y logra mejoras notables respecto de soluciones provenientes de resultados de otras estrategias. Por lo tanto, ciertamente se puede utilizar en el problema RDCMST,sea como herramienta primaria o complementaria. Además, los resultados parecen mostrar que el rendimiento de la implementación mejora a medida que crece la instancia, lo cual es promisorio, ya que en este proyecto se ha trabajando con una red de abstracción más pequeña que la real. Además, Giraph está construido sobre un sistema altamente escalable –Hadoop–, en el cual fácilmente, con solo agregar más nodos de procesamiento al clúster, es posible aumentar la potencia de procesamiento y la capacidad de memoria.

Por cuestión de tiempo, no se realizaron experimentos modificando la cantidad de nodos procesadores, algo que daría información valiosa sobre la escalabilidad de la implementación desarrollada; sin embargo, no hay ninguna razón para pensar que ella no pueda manejar instancias mucho más grandes.

Finalmente, hubiera sido ideal evaluar la estrategia utilizando instancias más reales de redes con diferentes configuraciones y buscar el número óptimo de trabajadores que se ocupan del conjunto de datos utilizado; sin embargo, obtener tales conjuntos de datos es realmente difícil. Sin embargo, la expectativa es abordar estos problemas a futuro.

Comentarios finales

Este proyecto perseguía dos grandes objetivos, uno derivado del otro. Primero, se buscaba un enfoque distribuido para resolver RDCMST, y se encontró el *framework* Apache Giraph; luego, se quiso evaluar qué tan rápido, simple y escalable es Giraph para implementar soluciones a problemas como

RDCMST, y se encontró que la búsqueda local, indicada en el Algoritmo 8, se puede implementar completamente usando Giraph y que la mayor parte de la lógica, incluidas las partes clave de la búsqueda, se implementan fácilmente con el núcleo del paradigma de programación centrado en vértices. Sin embargo, algunos detalles, relacionados con la conservación del estado, necesita características más avanzadas de Giraph, lo que implica dificultades adicionales en el diseño del algoritmo. En este proyecto se confirmó que escalar el tamaño de entrada para el problema de RDCMST no tuvo ningún impacto en el desarrollado, las ejecuciones de código muestran resultados aceptables, incluso en instancias que podrían no ser lo suficientemente grandes como para explotar el potencial de Giraph.

CONCLUSIONES Y TRABAJO FUTURO

Con este proyecto se presenta la primera estrategia distribuida conocida para resolver el problema RDCMST, basada en la idea de una exploración de vecindad paralela. La implementación de esta estrategia en una arquitectura de software distribuida permite lidiar con instancias problemáticas de decenas de miles de vértices, un tamaño que no se ha reportado hasta ahora.

En el proyecto se diseñó, implementó y presentó una colección de algoritmos que sirven para resolver problemas comunes en el diseño de algoritmos de grafos utilizando el paradigma de programación vértice-céntrica y, en consecuencia, una arquitectura distribuida. Los algoritmos desarrollados se pueden utilizar en procedimientos en los que puede haber mutaciones en grafos dirigidos, donde se quiera realizar un seguimiento de la posición de cada vértice o de cualquier variable relacionada con él. El diseño de estos algoritmos fue el trabajo más duro de este proyecto, por lo que se espera que ellos representen una contribución valiosa.

El paradigma de programación centrado en vértices brindó una granularidad de tareas flexible para distribuir la solución del problema, basado en el hecho de que cada vértice tiene que computar sus operaciones de búsqueda micro local. Por su parte, el *framework* de código abierto Apache Giraph ofreció una arquitectura distribuida donde ejecutar el algoritmo mediante la explotación de recursos computacionales, sin embargo, aunque Giraph permitió resolver grandes instancias del problema, es claro que penaliza la ejecución en instancias pequeñas, como las que se han probado en enfoques anteriores.

Los resultados sugieren el potencial de la solución alcanzada para manejar casos de problemas más grandes que los que fueron evaluados, pero también muestra su debilidad de rendimiento para resolver las instancias de problemas que se han validado en la literatura, que son demasiado pequeñas para aprovechar un enfoque distribuido. Sin embargo, esta limitación es común a casi todas las demás soluciones implementadas en este tipo de *frameworks* distribuidos; por lo tanto, estos resultados se centran en qué tan bien se pueden resolver grandes instancias, en lugar de comparar el rendimiento de este enfoque con enfoques anteriores (a pesar de que se verificó que esta implementación produjo soluciones correctas).

Se mostró además cómo los resultados de rendimiento obtenidos se afectaron con la estructura de la solución inicial sencilla propuesta, el alto grado de vértice de la raíz en dicha estructura redujo el nivel paralelo que el algoritmo puede alcanzar en sus iteraciones iniciales. En consecuencia, se siguieron dos estrategias para obtener soluciones iniciales más alargadas y con ello, además de las mejoras de rendimiento, se confirmó que las implementaciones desarrolladas podrían usarse como una herramienta complementaria para refinar soluciones que ya son buenas.

Más allá de eso, a pesar de que fue posible lidiar con casos de problemas de decenas de miles de vértices, no fue posible establecer con certeza los límites reales de la solución alcanzada. Las restricciones de tiempo de este proyecto hicieron imposible experimentar con instancias más grandes para verificar su rendimiento en ellas, no obstante, la contribución del proyecto va más allá de la implementación, ya que la estrategia propuesta se puede codificar y desplegar en otras arquitecturas que pueden incluir enfoques de memoria compartida, incluso en *frameworks* distribuidos alternativos que utilizan el modelo de programación centrado en vértices [24]. Diferentes implementaciones podrían ofrecer mejor o peor rendimiento, según las instancias del problema.

Obtener una solución distribuida de RDCMST "queda" muy cerca de obtener una solución distribuida para EDRDCMT, cuyo algoritmo de solución secuencial inspiró el enfoque utilizado en el proyecto. El enfoque del trabajo realizado podría ayudar a resolver generalizaciones o derivaciones de RDCMST.

Como trabajo futuro, se recomienda evaluar a fondo la implementación realizada, teniendo en cuenta su escalabilidad y explorando sus límites reales, para lo que sería ideal poder utilizar las redes LR-PON originales de algunos

países europeos. Asimismo, sería interesante explorar algunas características adicionales de Giraph, como la partición de grafos personalizados, para mejorar el rendimiento e implementar la estrategia utilizando otras arquitecturas de software distribuidas –y medir las diferencias de rendimiento en cuanto al tamaño del problema–, para así poder hacer un análisis completo de los pro y los contra de las diversas implementaciones.

REFERENCIAS

[1] A. Arbelaez, D. Mehta, B. O'Sullivan, y L. Quesada, "A constraint-based local search for edge disjoint rooted distance-constrained minimum spanning tree problem," en *Integration of AI and OR Techniques in Constraint Programming: 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings* (LNCS, vol. 9075), Cham, Suiza: Springer, 2015. pp. 31–46.

[2] K.G. Murty, *Linear programming*. Hoboken, NJ: Wiley, 1983.

[3] C. Loaiza, "Distributed local search based on a parallel neighborhood exploration for the rooted distance-constrained minimum spanning-tree problem," tesis de maestría, Fac. Ingeniería, Univ, Icesi, Cali, Colombia, 2019.

[4] J. Eisner. (1997). State-of-the-art algorithms for minimum spanning trees: A tutorial discussion. [Online]. Disponible: https://www.cs.jhu.edu/~jason/papers/eisner.mst-tutorial.pdf

[5] I. Pyo, J. Oh, y M. Pedram, "Constructing minimal spanning/steiner trees with bounded path length," en *Proceedings ED&TC European Design and Test Conference* 1996, pp. 244–249.

[6] L. Gouveia, A. Paias, y D. Sharma, "Modeling and solving the rooted distance-constrained minimum spanning tree problem," *Computers & Operations Research*, vol. 35, no. 2, pp. 600–613, 2008.

[7] G. Dahl, L. Gouveia, y C. Requejo, "On formulations and methods for the hop-constrained minimum spanning tree problem," en *Handbook of optimization in telecommunications*, M. Resende y P. Pardalos, Eds. New York, NY: Springer, 2006, 493–515.

[8] H.F. Salama, D.S. Reeves, y Y. Viniotis, (1996). "An efficient delay-constrained minimum spanning tree heuristic," en *Proceedings of the 5th International Conference on Computer Communications and Networks*, 1996.

[9] M. Ruthmair y G. R. Raidl, "A kruskal-based heuristic for the rooted delay-constrained minimum spanning tree problem," en *Computer Aided Systems Theory - EUROCAST 2009* (LNCS, 5717), R. Moreno-Díaz, F. Pichler, y A. Quesada-Arencibia, Eds. Berlín-Heidelberg, Alemania: Springer, 2009, pp. 713–720.

[10] M. Berlakovich, M. Ruthmair, y G. R. Raidl, "A multilevel heuristic for the rooted delay-constrained minimum spanning tree problem," en *Computer Aided Systems Theory – EUROCAST 2011* (LNCS, 6927), R. Moreno-Díaz, F. Pichler, y A. Quesada-Arencibia, Eds. Berlin-Heidelberg, Alemania: Springer, 2012, pp. 256–263.

[11] M. Ruthmair y G. R. Raidl, "Variable neighborhood search and ant colony optimization for the rooted delay-constrained minimum spanning tree problem," en *Parallel Problem Solving from Nature, PPSN XI* (LNCS, vol. 6239), R. Schaefer, C. Cotta, J. Kołodziej, y G. Rudolph, Eds. Berlin-Heidelberg, Alemania: Springer, 2010, pp. 391–400.

[12] M. Ruthmair y G. R. Raidl, "A memetic algorithm and a solution archive for the rooted delay-constrained minimum spanning tree problem," en *Computer Aided Systems Theory – EUROCAST 2011* (LNCS, 6927), R. Moreno-Díaz, F. Pichler, y A. Quesada-Arencibia, Eds. Berlin-Heidelberg, Alemania: Springer, 2012, pp. 351-358

[13] R. Battiti, M. Brunato, y F. Mascia, *Reactive search and intelligent optimization* (Operations Research/Computer Science Interfaces, vol. 45), Cham, Suiza: Springer, 2008.

[14] M. Litoiu, M. Shaw, G. Tamura, N. M. Villegas, H. A. Muller, H. Giese, R. Rouvoy, y E. Rutten, "What can control theory teach us about assurances in self-adaptive software systems?," en *Software Engineering for Self-Adaptive Systems III* (LNCS, vol. 9640), R. de Lemos, D. Garlan, C. Ghezzi, y H. Giese, Eds. Cham, Suiza: Springer, 2017, pp. 90–134.

[15] G. Tamura, R. Casallas, A. Cleve, y L. Duchien, "QoS contract preservation through dynamic reconfiguration: A formal semantics approach," *Science of Computer Programming (SCP)*, vol. 94, no. 3, pp. 307-332, 2014.

[16] G. Tamura, "QoS-CARE: A reliable system for preserving QoS contracts through dynamic reconfiguration," tesis de doctorado, Université de Lille, Francia / Universidad de los Andes, Bogotá, Colombia, 2012.

[17] J. C. Muñoz-Fernández, G. Tamura, R. Mazo, y C. Salinesi, "Towards a requirements specification multi-view *framework* for self-adaptive systems," en 2014 XL Latin American Computing Conference (CLEI), Montevideo, 2014, doi: 10.1109/CLEI.2014.6965158.

[18] T. White, *Hadoop: The definitive guide*, Sebastopol, CA: O'Reilly Media, 2015.

[19] C. Martella, R. Shaposhnik, D. Logothetis, y S. Harenberg, *Practical graph analytics with apache giraph*. Berkeley, CA: APress, 2015.

[20] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, y G. Czajkowski, "Pregel: A system for large-scale graph processing," en *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp.135-146,

[21] S. Sakr, F. M. Orakzai, I. Abdelaziz, y Z. Khayyat, *Large-scale graph processing using Apache Giraph*, Cham, Suiza: Springer, 2016.

[22] M. McCool, J. Reinders, y A. Robison, *Structured parallel programming: pat- terns for efficient computation*, Waltham, MA: Elsevier, 2012.

[23] Spain Fyber-optic LRPON network [dataset]

[24] S. Heidari, Y. Simmhan, R. N. Calheiros, y R. Buyya, "Scalable graph processing *framework*s: A taxonomy and open challenges," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, Art. 60, 2018.

ANEXO I

Descripción gráfica del movimiento completo

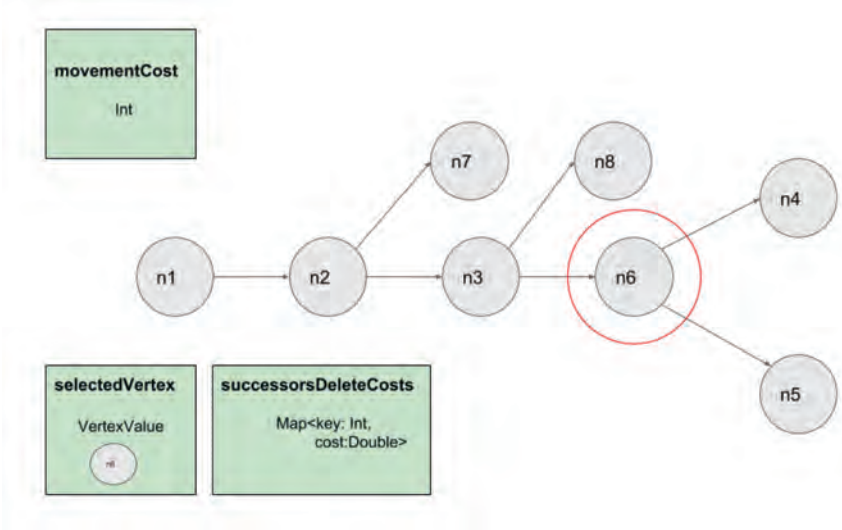
Este adjunto se preparó para ilustrar la solución completa basada en Giraph para resolver RDCMST e incluye un movimiento completo con todos sus escenarios. Antes de revisar las imágenes, caben las siguientes consideraciones.

Para empezar, hay dos tipos de etapas dentro de un movimiento, ambas tienen diferentes entornos de ejecución; para cada fase se muestran dos clases de imágenes, una para la etapa computation de superpasos, que se ejecuta en paralelo por los trabajadores –y a menudo requiere varias imágenes por fase–; la otra para la etapa master computation, que es ejecutada secuencialmente por el nodo maestro –y siempre requiere una sola imagen–.

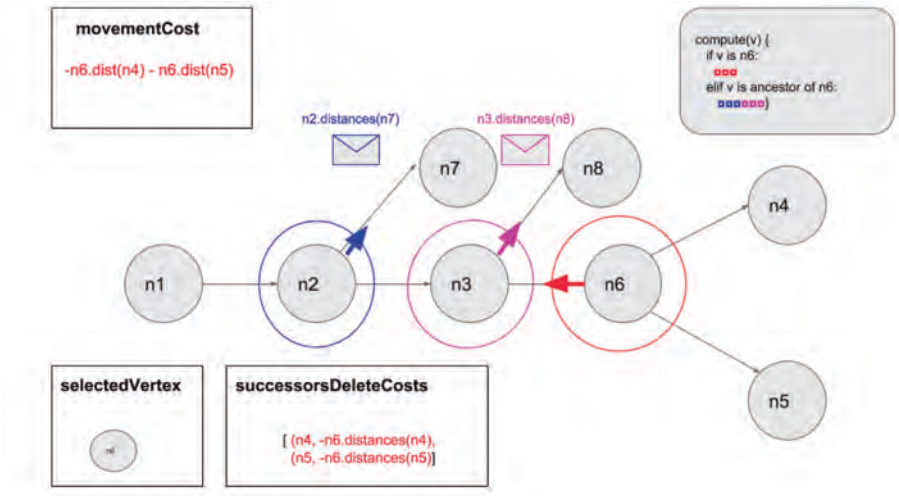
La etapa master computation solo se ocupa del estado de las variables globales: las variables emitidas, reductores y agregadores, las cuales en estas ilustraciones se tratan sin diferenciación y se presentan como rectángulos con esquinas cuadradas dibujadas en negro, en cuyo fondo se ilustra su estado actual. Si su fondo es verde, la variable se está inicializando, lo que solo puede suceder en la etapa master computation (además, la inicialización de una variable siempre incluye su tipo además de su valor inicial).

En las etapas computation de los superpasos, algunos vértices están rodeados de colores, esto significa que se ejecuta su función compute; en ocasiones se dibuja un rectángulo de esquina redondeada con el mismo color que contiene el pseudocódigo más significativo de la función. Cuando la función compute del vértice está escribiendo una variable global, los cambios producidos por ella se pintan con el mismo color dentro del rectángulo de la variable. Por ejemplo, en la Figura b el vértice *n6* escribe el costo de movimiento en el superpaso *0*. Adicionalmente, las imágenes muestran los mensajes salientes como sobres con su contenido escrito al lado. Finalmente, un gran rectángulo negro con su esquina redondeada –siempre ubicado en la esquina superior derecha de computation de los superpasos–, muestra el pseudocódigo que define cómo se asignan los diferentes colores a cada vértice en el gráfico.

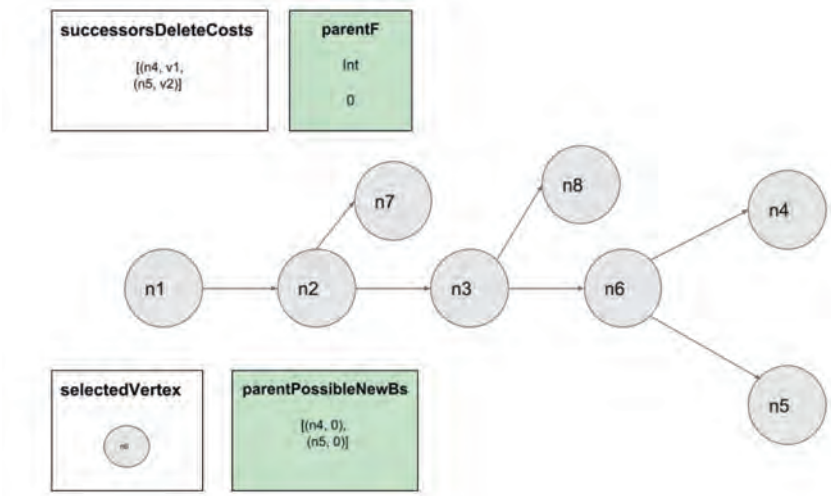
Como se dijo al inicio, estas imágenes se prepararon como una herramienta complementaria que ayude a comprender la descripción hecha de la solución completa basada en Giraph para resolver RDCMST, son entonces una ilustración gráfica de los algoritmos ahí presentados y no imágenes que puedan ser entendidas por sí mismas.



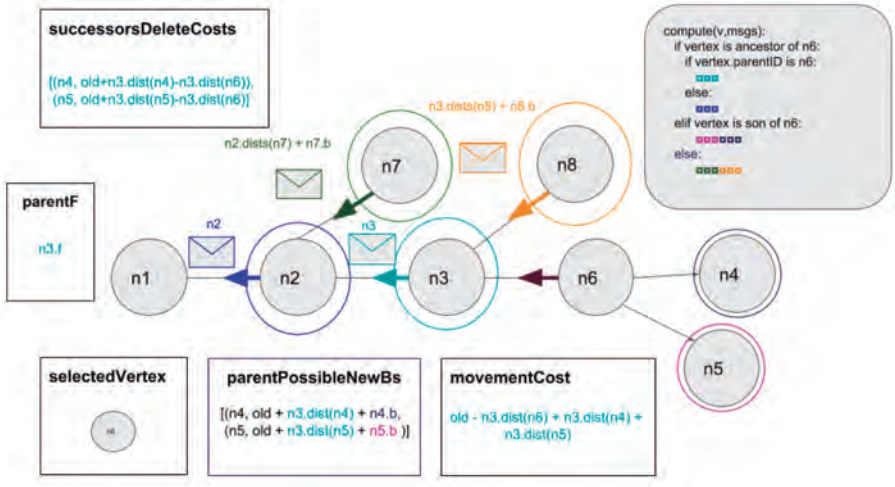
(a) Master Compute 0



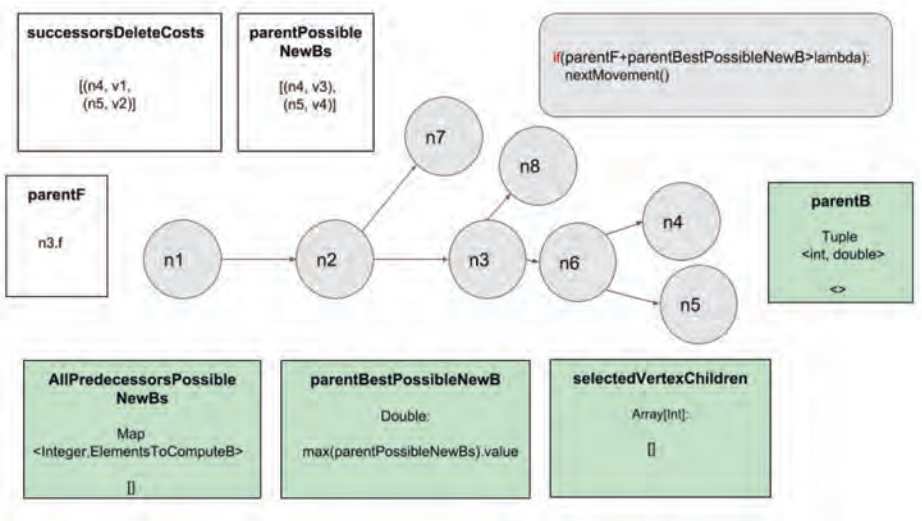
(b) Superpaso 0



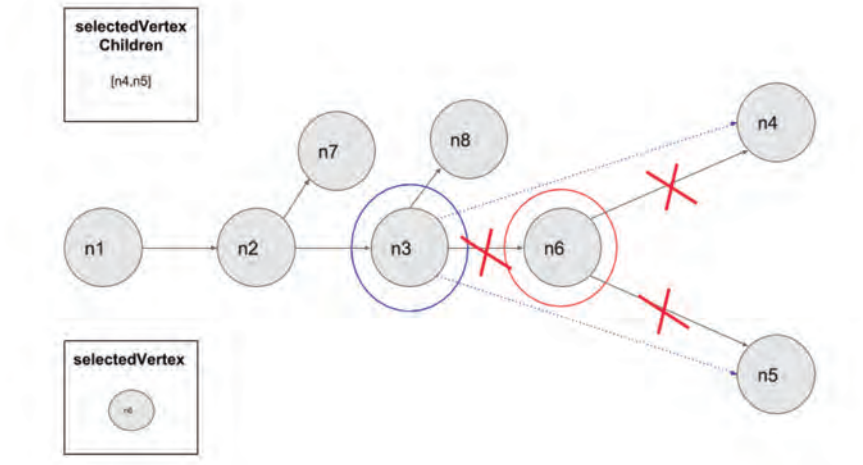
(**c**) **Master Compute 1**



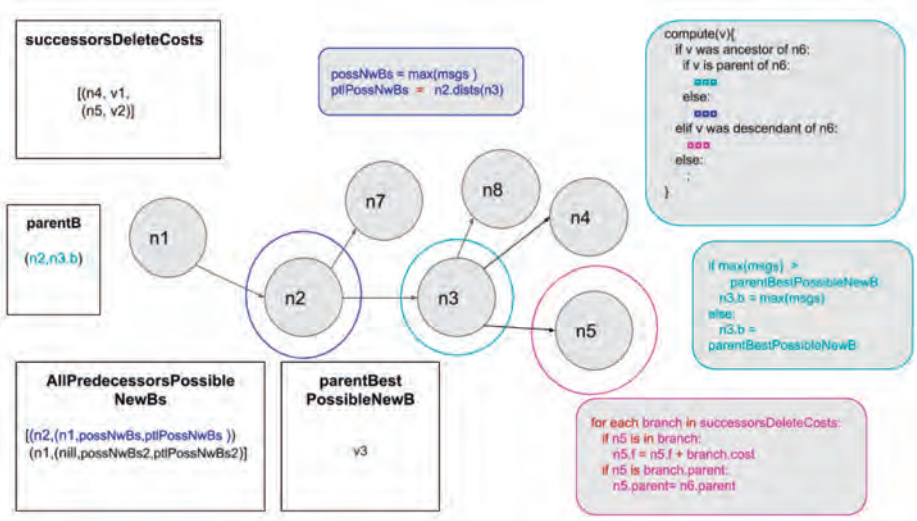
(**d**) **Superpaso 1**



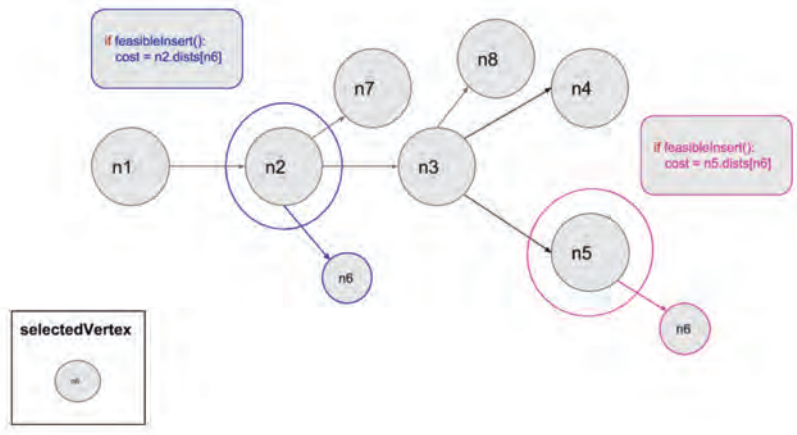
(e) Master Compute 2



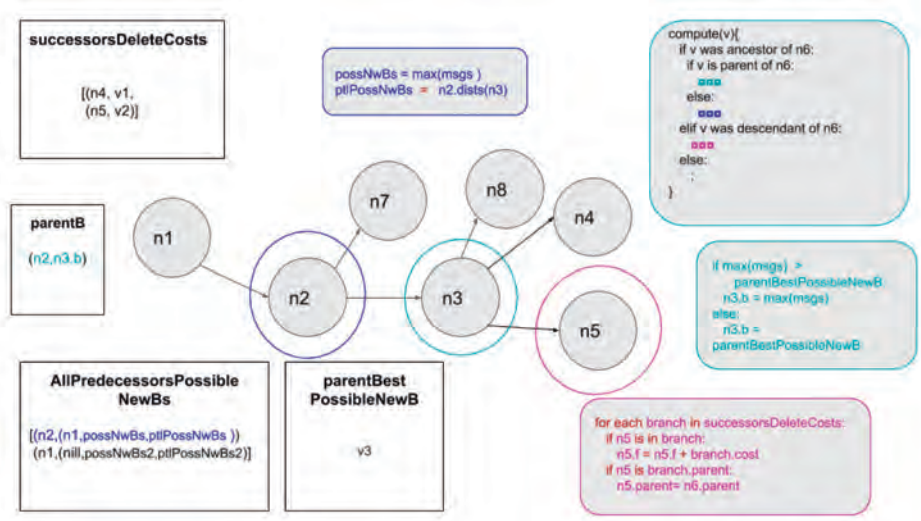
(f) Superstep 2-1



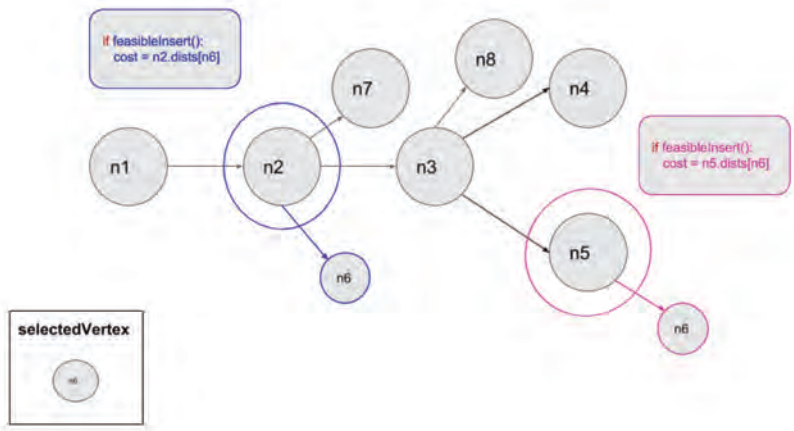
(g) **Superstep 2-2**



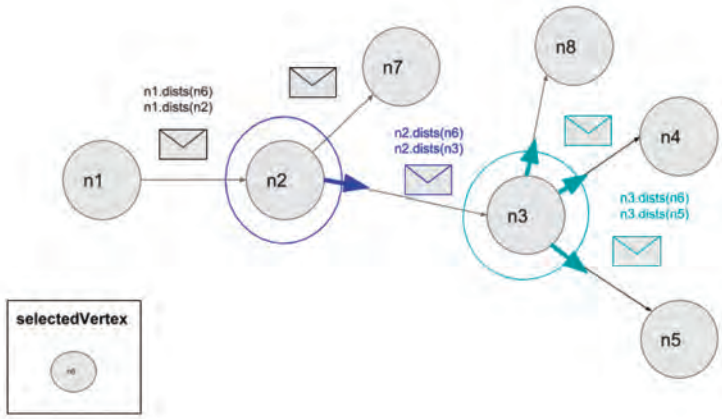
(h) **Superstep 2-3**



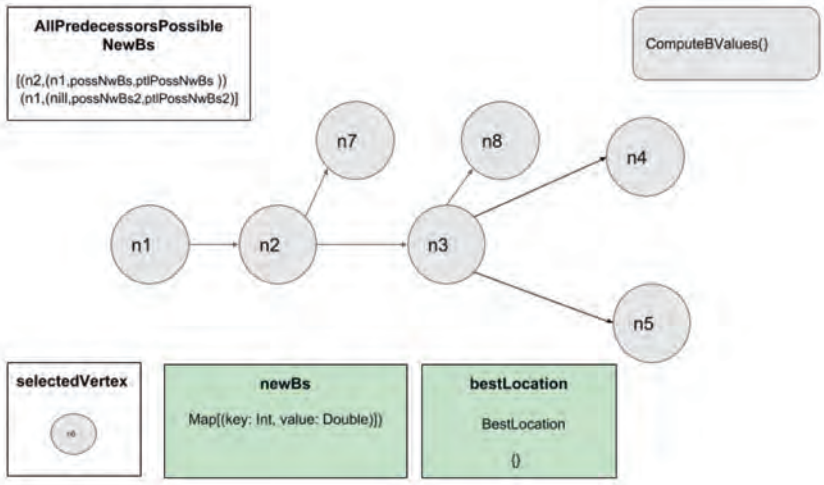
(i) Superpaso 2-2



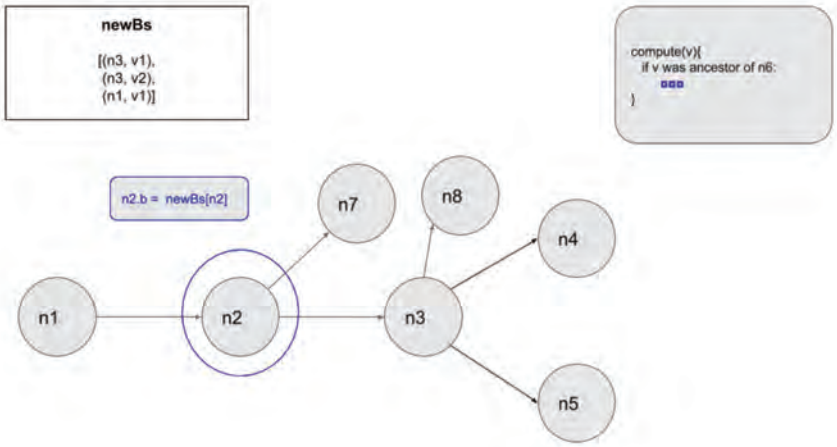
(j) Superpaso 2-3



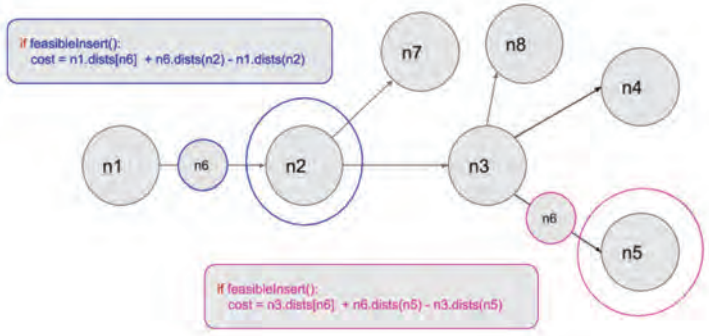
(k) Superpaso 2-4



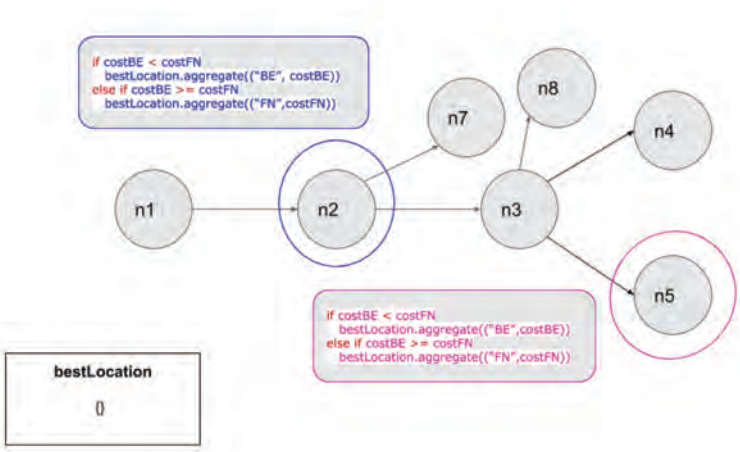
(l) Master compute 3



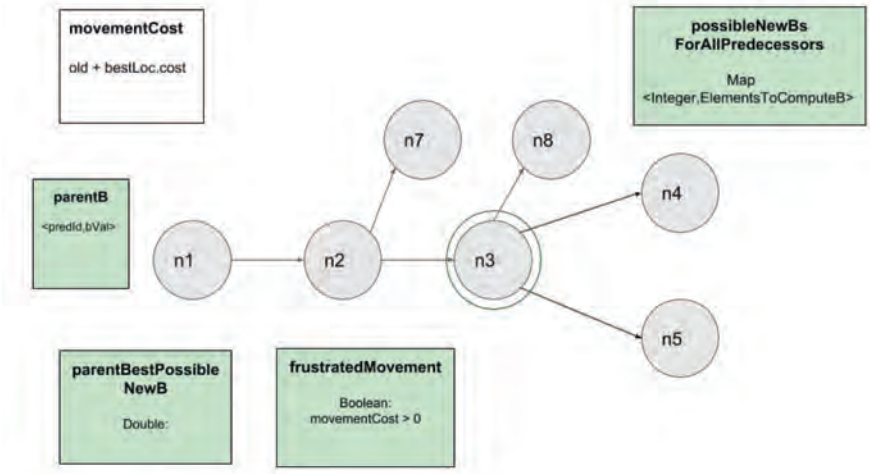
(**m**) **Superpaso 3.1**



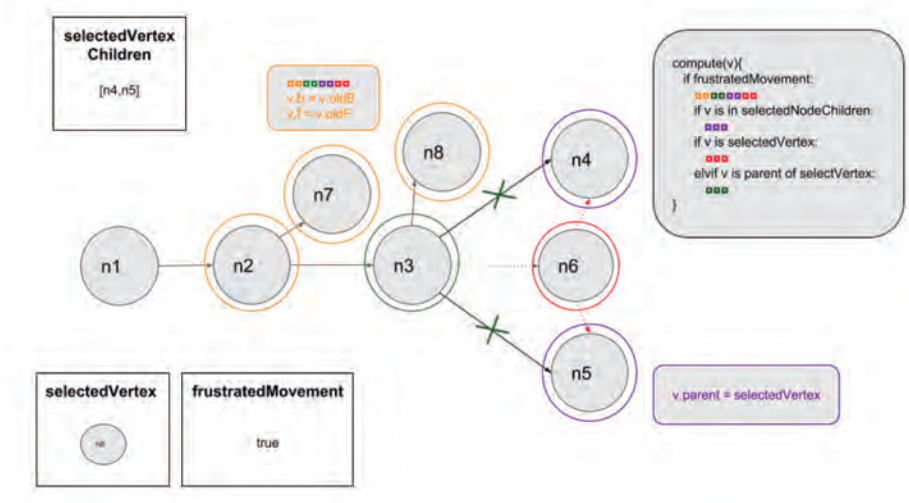
(**n**) **Superpaso 3-2**



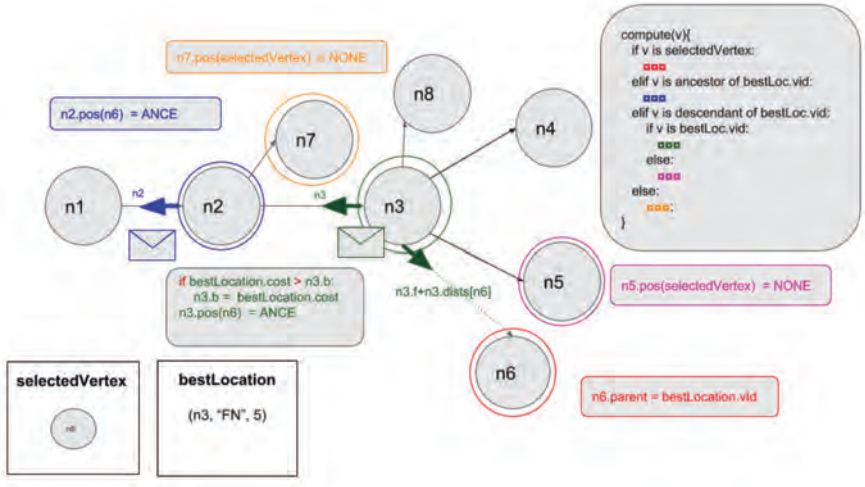
(o) **Superpaso 3-3**



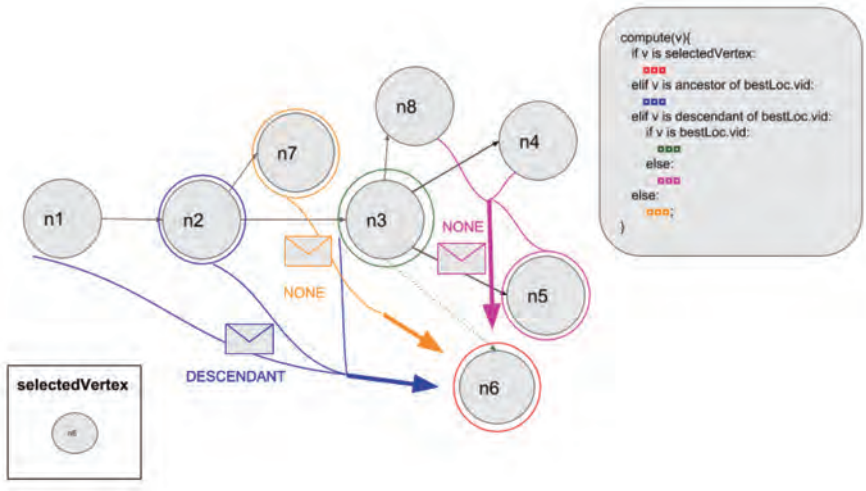
(p) **Master compute 4**



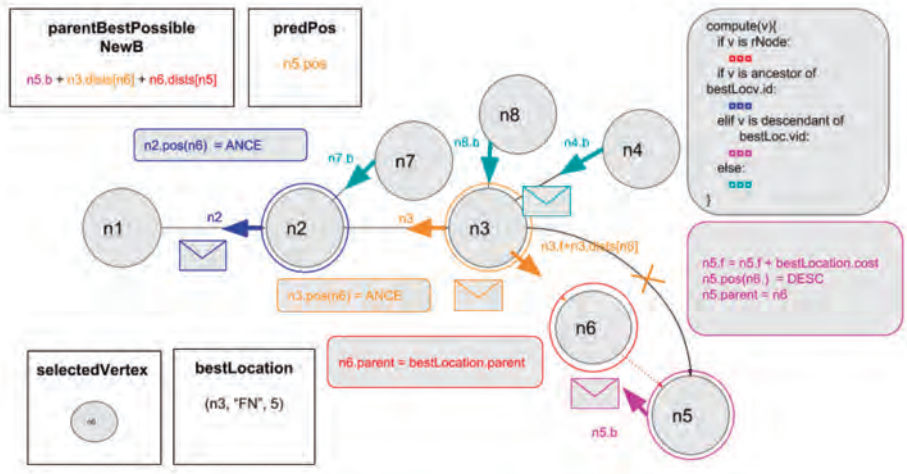
(q) **Superpaso 4-1**



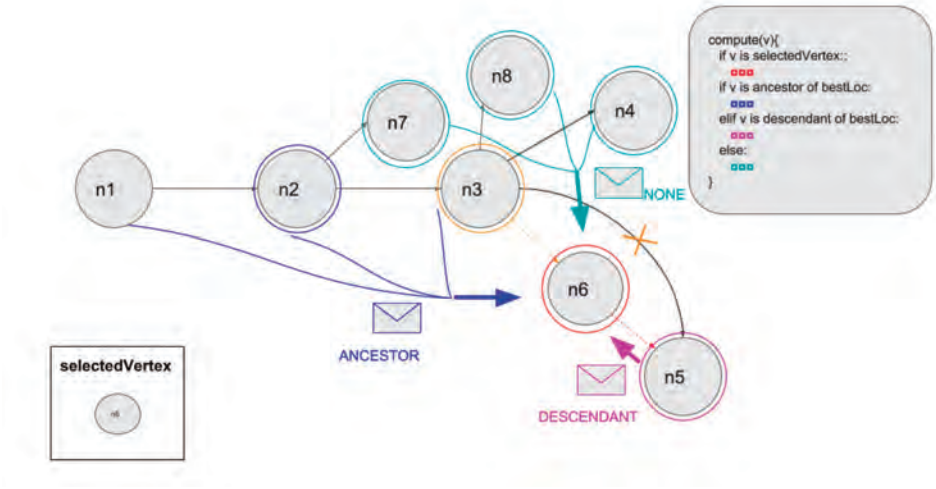
(r) **Superpaso 4-2**



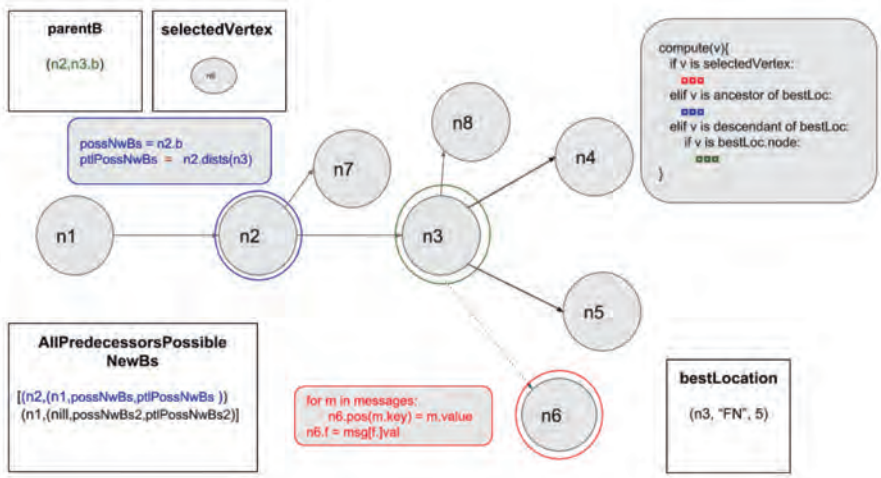
(s) **Superpaso 4-3**



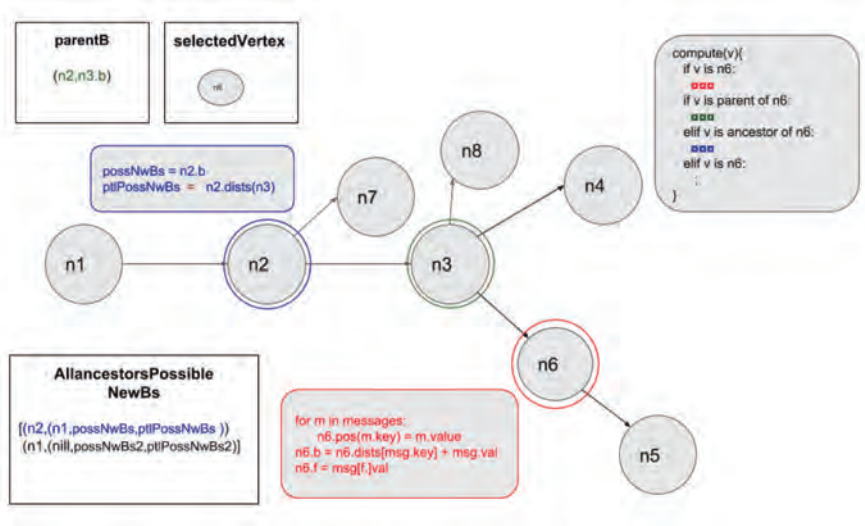
(t) **Superpaso 4-4**



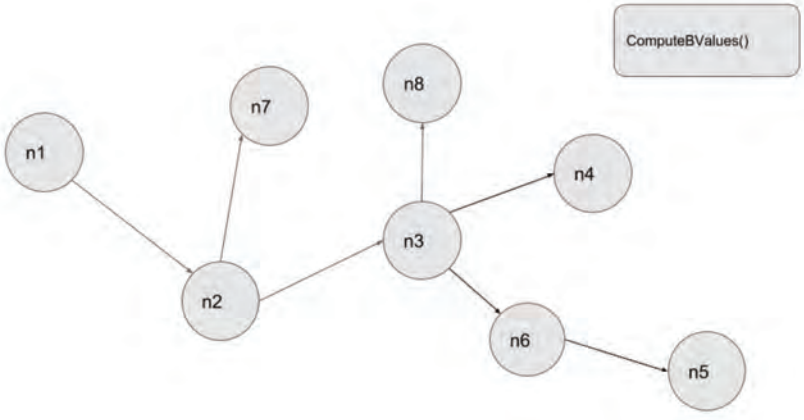
(u) **Superpaso 4-5**



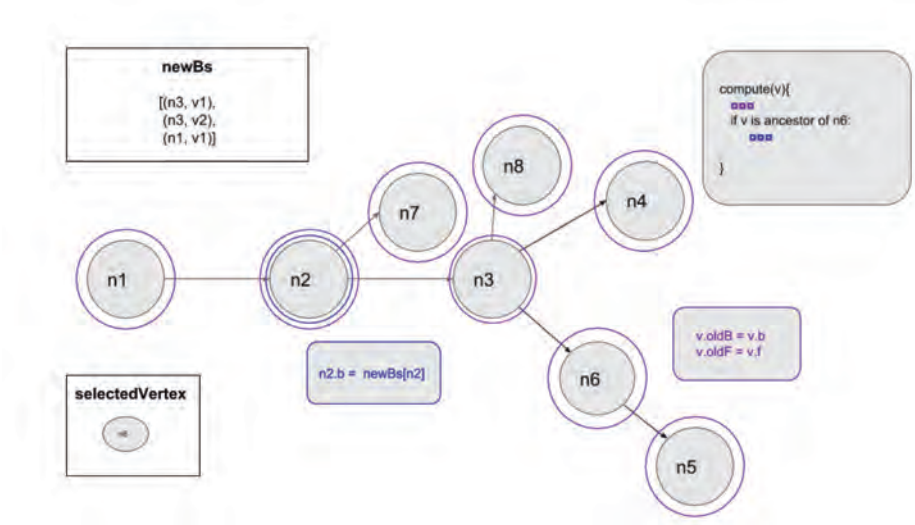
(v) **Superpaso 0-1**



(w) Superpaso 0-2



(x) Master compute 1



(y) **Superpaso 1**

La preparación de este libro estuvo al cuidado de Claros Editores S.A.S., finalizó en junio de 2020. En su preparación, realizada desde la Editorial Universidad Icesi, se emplearon los tipos: Baskerville MT Std de 9, 10 y 12 puntos; Book antigua de 8 puntos; Cambria Math de 12 puntos; Gill Sans MT de 8, 9, 10, 14, 19, 26 puntos; y Times New Roman de 10 puntos.

En la colección "Bitácoras de la Maestría" se presentan los resultados de las investigaciones base del desarrollo de tesis meritorias de la Universidad Icesi, en este caso se incluyen tres trabajos provenientes de la Facultad de Ingeniería, específicamente de la Maestría en Informática y Telecomunicaciones. El primero de ellos aborda tangencialmente la cuarta revolución industrial y se enfoca en la transformación de los modelos de negocio y operativo en una empresa de base tecnológica, hacia esquemas adaptables y flexibles que le permitan una entrega de valor continua, en plazos muy cortos, desde un modelo enmarcado en los valores y principios del desarrollo ágil. El segundo tiene relación con la seguridad de la información en dispositivos móviles y se enfoca en mejorar las tasas de detección de malware a través de un sistema de aprendizaje de máquina basado en redes neuronales artificiales para la clasificación de malware en Android. El tercero ofrece una solución al problema de optimización del árbol de recubrimiento mínimo restringido por la distancia raíz (RDCMST, Rooted Distance-Constrained Minimum Spanning Tree), con una característica especial, resuelve las carencias presentes en soluciones logradas en investigaciones previas: la ausencia de aproximaciones paralelas diseñadas para aprovechar varias unidades de procesamiento y la limitación a instancias de unos pocos miles de vértices.

UNIVERSIDAD ICESI