Capítulo 11
Monitoreo de indicadores de rendimiento en cadenas *Blockchain Ethereum*

Juan Camilo Giraldo Mejia - Jgiraldo1@tdea.edu.co
Julián Grisales Z. - Juliangris@gmail.com
Mauricio Amariles Camacho - Mauricio.amariles@tdea.edu.co
Tecnológico de Antioquia

I. INTRODUCCIÓN

Blockchain es una tecnología que cambia el concepto de la confianza entre las personas, ya que resuelve el problema de validar la autenticidad de los datos sin requerir un intermediario de confianza, lo que permite que cualquier miembro de la red verifique la validez de los datos de forma independiente, por medio de algoritmos criptográficos que autorizan y registran una serie de transacciones que se ejecutan y registran sobre todo el sistema [1]. Esta tecnología permite a gobiernos, tecnologías financieras (*Fintech*), empresas emergentes e individuos agregar a sus procesos: registro, trazabilidad, seguridad y almacenamiento distribuido de sus datos [2].

Una de las tecnologías más destacadas de *Blockchain* es Ethereum, ya que desde noviembre de 2018 sus aplicaciones contaban con una cuota de mercado estimada en US$21.447.799.407, ubicándose en segundo lugar por debajo de Bitcoin [3].

Lo que hace tan atractiva esta tecnología es que su base es de código abierto, en donde una gran cantidad de personas o empresas podrían beneficiarse aumentando la seguridad, mejorando sus procesos y reduciendo costos de manera considerable. Sin embargo, *Ethereum* todavía tiene una serie de desafíos técnicos referentes a la privacidad, escalabilidad y rendimiento [4].

Habilitar *Ethereum* puede llevar tiempo por la falta de conocimiento en todos los elementos que hacen parte del sistema, y por la ausencia de herramientas y modelos confiables y de fácil acceso que permitan identificar anomalías que afecten el rendimiento.

En investigaciones como la que se presenta en [5], se afirma que los modelos actuales de monitoreo generan sobrecarga en el sistema *Blockchain,* ofrecen pocos parámetros en el monitoreo y carecen de capacidad de escalabilidad.

En [6] se ilustra cuáles son los componentes y variables que generan los mayores cuellos de botella en el sistema, como son la velocidad de procesamiento, la memoria de uso, la velocidad de respuesta entre nodos y el consumo de energía. En [7], por su parte, se demuestra que el rendimiento también puede verse afectado por ataques informáticos que puede sufrir el sistema como está construido actualmente.

El presente trabajo busca identificar y caracterizar las métricas del sistema *Blockchain Ethereum,* y con base en esta información se propone un modelo de monitoreo en relación con el rendimiento en un caso aplicado.

En la actualidad, la adopción de sistemas que se ejecutan sobre redes privadas *Blockchain Ethereum* viene en aumento, debido a que es una plataforma de software abierta con características de descentralización, inmutabilidad y trazabilidad, que permite la creación de aplicaciones en diferentes ámbitos, sin embargo, al igual que otras arquitecturas, este sistema está soportado por múltiples componentes de hardware como: CPU, memoria RAM, memoria ROM y la tarjeta de red, los cuales, al presentar saturación o fallas, provocan cuellos de botella en el sistema *Blockchain*. Estas saturaciones pueden ser provocadas por la variación de parámetros como: tamaño de bloque, política de respaldo, latencia en la red, asignación de recursos, etc. Estos eventos ocasionan ineficiencia y conflictos en el sistema [8].

Se ha evidenciado que a medida que el sistema crece, adicionando nuevos nodos a la red, el algoritmo de consenso genera también cuellos de botella, ya que todos los nodos de la red deben dar el consenso para aprobar las transacciones que se ejecutan [9].

II. MATERIALES Y MÉTODOS

Blockchain fue presentado por primera vez en el año 2008 por el autor con pseudónimo Satoshi Nakamoto, en el tratado llamado *Bitcoin* [10], en el cual propone un sistema de dinero electrónico definido como criptomoneda, conformado por múltiples nodos, soportados en una red punto a punto, donde cada uno de los nodos se encuentra ubicado en diferentes sitios [11].

En su propuesta, Nakamoto describe los inconvenientes de un sistema de pago centralizado donde se depende de una única entidad de confianza, la cual se encarga de procesar y almacenar las transacciones en el sistema de pagos,

y por la condición de estar soportado sobre un esquema centralizado, puede presentar problemas como costo de transacción, eficiencia y seguridad [10]. Lo anterior lleva a que la tecnología *Blockchain* pueda ser utilizada como base de datos, contratos inteligentes y transacciones condicionales [12].

En enero de 2018, se realizó una medición en la velocidad de procesamiento de transacciones por sistemas de pago tradicionales, en comparación con los sistemas de pagos no tradicionales soportados sobre redes *Blockchain* como *bitcoin* y *Ethereum*. Se concluyó que los sistemas de pagos tradicionales como Visa son más eficientes que los sistemas de pago no tradicionales, en lo referente a la velocidad en la que se procesan las transacciones.

Al evidenciar esta problemática, diferentes autores han profundizado en el análisis de las métricas que inciden directamente en el rendimiento del procesamiento de transacciones en las redes *Blockchain*, y han identificado cuellos de botella en diferentes elementos.

La red, conformada por múltiples nodos, generalmente localizados en diferentes ubicaciones geográficas, tiene una métrica llamada latencia, la cual incide directamente en el rendimiento de las transacciones, razón por la cual se han propuesto simuladores que modelen los componentes de bloque, nodo y características de la red para predecir el comportamiento relacionado con el rendimiento [13].

Adicionalmente, algunas investigaciones han determinado que la base de datos debe ser modelada a nivel local, nivel remoto y nivel distribuido, ya que en cada punto tiene mayor o menor incidencia en el rendimiento del sistema en métricas, como latencia de escritura y latencia de lectura [14].

El algoritmo de consenso en redes *Blockchain* cumple un papel fundamental en el correcto funcionamiento del protocolo, porque es el que permite aprobar transacciones con el consenso de los nodos. Precisamente, el tiempo que tardan los nodos en dar el consenso para aprobar las transacciones incide en el rendimiento del sistema, donde los componentes de CPU, memoria y red desempeñan un papel fundamental en la respuesta efectiva del consenso de cada nodo [15].

La memoria RAM en *Blockchain* es la encargada de almacenar las transacciones que posteriormente serán procesadas en el sistema. Este componente funciona con esquemas de encolamiento, por lo tanto, su desempeño incide considerablemente en el rendimiento del nodo.

Las métricas que inciden en el

rendimiento son las transacciones por bloque, el tiempo de minado en cada bloque, transacciones por segundo, *pool* de memoria, tiempo de espera en memoria, número de transacciones sin confirmar, número total de transacciones y número de bloques generados [16].

En los sistemas *Blockchain*, la CPU procesa las transacciones en los nodos, y es la encargada del proceso de minado en redes *Ethereum,* ya que realiza una de las tareas más críticas en el rendimiento de transacciones del sistema. Algunos autores proponen evaluar su desempeño de procesamiento por medio de la ejecución de bloques consecutivos de transacciones sintéticas, que permitan ajustar los valores para alcanzar el máximo desempeño [12].

En algunas aplicaciones, como los contratos inteligentes en *Blockchain Ethereum,* es necesario firmar digitalmente las transacciones para garantizar la seguridad entre el remitente y el destinatario de la transacción. Esta propiedad afecta el rendimiento de las transacciones del sistema, ya que incide en la métrica de tiempo de propagación de bloque [17].

Al identificar los elementos que generan los principales cuellos de botella en el sistema en cuanto al rendimiento de transacciones, nace la necesidad de monitorear diferentes métricas y establecer un modelo que permita medir y analizar detalladamente las diferentes etapas del proceso de generación de la cadena de bloques.

Para el monitoreo de los indicadores, se utilizaron dos equipos con sistema operativo Linux, se instala cliente *Geth Ethereum* (Aplicación oficial *Ethereum*), *Json* con los datos del bloque, y el administrador de redes privadas de *Ethereum Puppeth.*

III. RESULTADOS

Se presentan los siguientes elementos para el monitoreo de indicadores de rendimiento *Blockchain* de *Ethereum*:

Colector de *logs*: corresponde a los registros requeridos en los diferentes componentes que se están generando correctamente, este elemento tiene la función de recolectarlos: colecta los logs en Red, los logs de Blockhain de Ethereum, y los registros del rendimiento del nodo.

Colector de registros en capa de red y nodo: la forma más simple de recolectar los registros en la capa de red y los componentes del nodo es por medio de subprogramas, APIS o sistemas de registros (*syslogs*), que consuman directamente los datos de los archivos de *logs* del sistema operativo o por medio de la invocación de comandos nativos que permitan extraer de forma segura los datos almacenados. Se propone extraer las

métricas en los componentes de red, CPU, memoria RAM, disco duro.

Colector de registros en capa *Blockchain Etherum:* para recolectar los registros del nodo *Blockchain Ethereum*, también existen APIS que nos facilitan el trabajo de recolección de estos registros, los cuales se conectan directamente sobre el nodo y ejecutan operaciones para extraer las métricas.

Almacenamiento de *logs* (normalizados y convertidos): este componente lo que busca es almacenar únicamente los registros relacionados con métricas de rendimiento de una forma estructurada y organizada, con el fin de que pueda ser analizada y cargada en componentes externos al sistema en donde, por ejemplo, se le haga una minería de datos a esta información.

Presentación WEB: una vez se obtienen los datos de rendimiento de métricas *Blockchain* de *Ethereum,* se requiere un componente web con gráficas enriquecidas que permita visualizar la variación en el tiempo de los diferentes indicadores, con el fin de entender el comportamiento del sistema y poder tomar decisiones cuando se presenten demoras o fallas en el mismo.

Analítica de datos para anomalías: contar con un repositorio de *logs* referentes a métricas de rendimiento para analizarlas en modelos de minería de datos, como las redes neuronales, y de árboles de decisión. Por lo tanto, se debe construir una secuencia de extracción, transformación y carga, conocida como ETL, en donde el destino sea una base de datos independiente del nodo *Ethereum*, para ejecutar modelos de entrenamiento con predicción y experimentos determinando valores de referencia en rendimiento, y que con base en la comparación de dos sets de datos el modelo tenga la capacidad de determinar anomalías y causas y muestre en tiempo real el desfase de rendimiento del sistema.

IV. PRUEBAS – FUNCIONALIDAD

A. Generador y colector de métricas de rendimiento

Paso 1: recolectar métricas de red, del procesamiento de la CPU, memoria RAM y ROM.

El ambiente de pruebas corresponde a dos nodos *blockchain* sobre máquinas virtuales Linux Ubuntu. En cada nodo está instalado el cliente *blockchain* (*geth*). Los nodos se unen en una red *blockchain* por medio de un archivo Génesis (*static-nodes.json).*

Para recolectar las métricas de los componentes de red, CPU, memoria y disco se hace uso del agente *open source telegraf,* el cual se instala como un plugin. Este tiene la capacidad

de recolectar métricas y eventos de aplicaciones en diferentes sistemas operativos.

Este colector de registros se instala en los dos nodos *Blockchain Ethereum* y se habilita el servicio: una vez se tiene el agente instalado y se está ejecutando como servicio, se indican en el archivo de configuración las métricas requeridas para recolectar.

```
1794 ###############################################################################
1795 #                            INPUT PLUGINS                                    #
1796 ###############################################################################
1797
1798
1799 # Read metrics about cpu usage
1800 [[inputs.cpu]]
1801   ## Whether to report per-cpu stats or not
1802   percpu = true
1803   ## Whether to report total system cpu stats or not
1804   totalcpu = true
1805   ## If true, collect raw CPU time metrics.
1806   collect_cpu_time = false
1807   ## If true, compute and report the sum of all non-idle CPU states.
1808   report_active = false
```

Fig. 1. *Configuración de las métricas en CPU*

```
1821 # Read metrics about disk IO by device
1822 [[inputs.diskio]]
1823   ## By default, telegraf will gather stats for all devices including
1824   ## disk partitions.
1825   ## Setting devices will restrict the stats to the specified devices.
1826   devices = ["sda", "sdb", "vd*"]
1827   ## Uncomment the following line if you need disk serial numbers.
1828   # skip_serial_number = false
```

Fig. 2. *Configuración de las métricas en disco*

```
1855 # Read metrics about memory usage
1856 [[inputs.mem]]
1857   # no configuration
1858
1859
1860 # Get the number of processes and group them by status
1861 [[inputs.processes]]
1862   # no configuration
1863
1864
1865 # Read metrics about swap memory usage
1866 [[inputs.swap]]
1867   # no configuration
```

Fig. 3. *Recolección de las métricas en memoria*

```
3762 # # Read metrics about network interface usage
3763  [[inputs.net]]
3764 #   ## By default, telegraf gathers stats from any up interface (excluding loopback)
3765 #   ## Setting interfaces will tell it to gather these explicit interfaces,
3766 #   ## regardless of status.
3767 #   ##
3768     interfaces = ["ens33", "lo"]
3769 #   ##
3770 #   ## On linux systems telegraf also collects protocol stats.
3771 #   ## Setting ignore_protocol_stats to true will skip reporting of protocol metrics.
3772 #   ##
3773 #   # ignore_protocol_stats = false
3774 #   ##
```

Fig. 4. *Recolección de métricas en Red*

Paso 2: se recolectan las métricas *Ethereum* vía API.

La recolección de las métricas del nodo *Ethereum* se obtiene por tres métodos: API, *Syslog,* Aplicación *Ethereum.* La red está conformada por dos nodos debido a las limitaciones de recursos en la máquina física.

Vía API: ejecutar servicio que extraiga métricas de monitoreo. Aquí hacemos uso de un API llamada *eth-net-intelligence-api,* donde debemos indicar los datos del nodo en el archivo *app.json* para recolectar los registros generados por el nodo *Ethereum.* Los parámetros que se deben diligenciar son:

– RPC_HOST (dirección IP del nodo donde se van a extraer los datos).

– RPC_PORT (puerto de conexión RPC para ejecutar comandos).

– LISTENING_PORT (puerto del nodo donde se van a extraer los datos).

– INSTANCE_NAME (identificador de la instancia cuando inicia el nodo).

– WS_SECRET (contraseña que se define para conectarse a un *front end* para graficar información).

– WS_SERVER (servidor donde se van a enviar los datos recolectados).

Al ejecutar el API podemos visualizar las métricas que extrae el nodo *Blockchain Ethereum:*

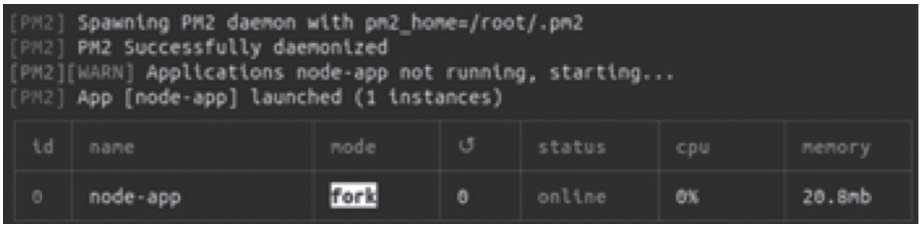

Fig. 5. *Métricas Blockchain Ethereum*

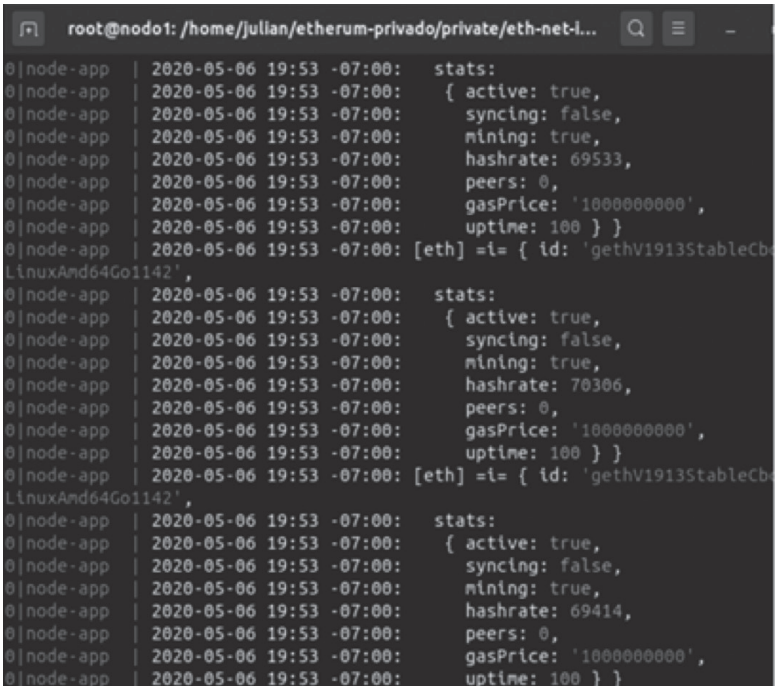

Fig. 6. *Extracción de métricas Ethereum vía API*

En la Figura 6 se presentan las métricas del gas consumido, el tiempo de validación de la transacción y el id del bloque.

Almacenamiento y visualización de métricas de rendimiento Blockchain Ethereum:

En el paso anterior, el API *eth-net-intelligence-api* extraía métricas de *Blockchain ethereum*. Existe un componente de capa de presentación (*front-end*) llamado *eth-netsats,* que consume esos registros y los grafica en una interfaz web: número de bloque, número de nodos (en este punto solo es un nodo), tiempo de último bloque, promedio de tiempo de bloque, promedio de latencia de la red, tiempo de bloque, dificultad, precio de GAS, Límite de GAS, propagación del bloque, últimos bloques minados, velocidad de minado, transacciones pendientes, bloque actual.

Almacenar métricas Blockchain Ethereum en bases de datos:

Para almacenar las métricas de Blockchain Ethereum en base de datos existen dos métodos:

- Vía *Syslog*
- Vía Aplicación *Ethereum*

Vía *Syslog:* el servicio *syslog* permite visualizar los registros del nodo en tiempo real, se puede visualizar con

el comando *tail-f/var/log/syslog|rep eth_blocknumber*: Este registro puede almacenarse en una base de datos *open source* llamada *influxdb*, por medio del servicio *telegraf:*

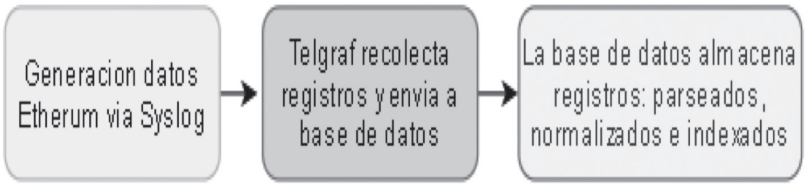

Fig. 7. *Flujo de datos Syslog hacia base de datos*

Para habilitar el almacenamiento en la base de datos, es necesario: Instalar un software llamado *influxdb,* crear una base de datos llamada *telegraf* con una tabla llamada *syslog,* crear un usuario para la aplicación con permisos de escritura y habilitar envío de registros desde *telegraf:*


```
104 # Configuration for sending metrics to InfluxDB
105 [[outputs.influxdb]]
106   ## The full HTTP or UDP URL for your InfluxDB instance.
107   ##
108   ## Multiple URLs can be specified for a single cluster, only ONE of the
109   ## urls will be written to each interval.
110   # urls = ["unix:///var/run/influxdb.sock"]
111   # urls = ["udp://127.0.0.1:8089"]
112   urls = ["http://127.0.0.1:8086"]
113   ## HTTP Basic Auth
114   username = "test"
115   password = "[redacted]"
```

Fig. 8. *Configuración telegraf para escritura de registros en influxDB*

Paso 3: visualización de métricas de red, CPU, memoria y disco

La aplicación *Influxdb* es compatible con la aplicación *Chronograf,* la cual tiene la capacidad de consumir las métricas de la base de datos y graficarlas en una interfaz web:

La aplicación permite ejecutar *querys* a la base de datos vía web y extraer cualquier registro que esté almacenado en esta. También es posible consultar vía web los eventos *syslog* y datos en la base de datos en tiempo real o tiempo pasado.

Paso 4: almacenar métricas de red, CPU, memoria y disco en la base de datos:

El almacenamiento se realiza desde el cliente *telegraf* con la configuración que anteriormente se había aplicado.

Fig. 9. Almacenamiento de métricas en red, CPU, memoria y disco en base de datos Influxdb

Paso 5: alertar eventos críticos y modelar detección de anomalías

Para alertar en tiempo real, hacemos uso de la herramienta *Kapacitor,* la cual es un analizador de *logs* en tiempo real y genera alertas preestablecidas de acuerdo con la parametrización. A continuación, se presenta alerta cuando la CPU está por encima del 80 % en relación con el consumo total.

V. CONCLUSIONES

En el nivel de monitoreo no se recomienda hacer uso del método RPC, ya que este induce en el sistema una sobrecarga adicional de procesamiento de los datos que puede impactar el rendimiento. Se propone monitorear el sistema por medio de la extracción de métricas en el nivel de aplicaciones colectoras de métricas y emplear el método *Syslog* para su análisis.

Almacenar todas las métricas en una base de datos centralizada permite tener trazabilidad sobre los indicadores de rendimiento y ayuda a entrenar el sistema bajo algoritmos de *machine learning* para detectar anomalías.

La validación con herramientas de código abierto permite la integración de nuevos componentes en el sistema para resolver necesidades específicas de rendimiento y monitoreo en un sistema *Blockchain Ethereum.*

Se proponen pruebas de evaluación de rendimiento del sistema, con el fin de que las organizaciones e investigadores pongan a prueba sus redes *Blockchain Ethereum,* ante eventos relacionados con degradación del servicio por la afectación del rendimiento en algunos componentes, y tengan la posibilidad de monitorearlos y predecirlos bajo el modelo propuesto.

En un servidor *Ethereum,* los elementos de CPU, memoria, disco y red tienen un alto impacto en el rendimiento de las transacciones.

VI. REFERENCIAS

[1] J. Mattila, "The Blockchain Phenomenon. The Disruptive Potential of Distributed Consensus Architectures. *ETLA Working Papers,* n.° 38, 2016.

[2] B. Jepkemei and A. Kipkebut, "Blockchain - A Disruptive Technology in Financial Assets", *IRE Journals,* vol. 2, n.° 9, pp. 38-47, 2019.

[3] A. S. Deshpande, *Design and Implementation of an Ethereum-like Blockchain Simulation Framework.* 2018.

[4] H. F. Leppelsack, "Experimental Performance Evaluation of Private Distributed Ledger Implementations" (Experimentelle Performanz Evaluierung von privaten Distributed Ledger Implementierungen), Master's thesis in Informatics, Technical University of Munich, April 2018.

[5] P. Zheng, Z. Zheng, X. Luo, X. Chen, and X. A. Liu, "A detailed and real-time performance monitoring framework for blockchain systems. In *Proceedings of International Conference on Software Engineering,* 2018, pp. 134-143. [On line]. https://doi.org/10.1145/3183519.3183546.

[6] H. Sukhwani, J. M. Martínez, X. Chang, K. S. Trivedi, and A. Rindos, "Performance modeling of PBFT consensus process for permissioned blockchain network (hyperledger fabric)". In *Proceedings of the IEEE Symposium on Reliable Distributed Systems,* Sept. 2017, pp. 253-255. [On line]. https://doi.org/10.1109/SRDS.2017.36

[7] F. Schussler, P. Nasirifard, H.-A. Jacobsen, "Attack and Vulnerability Simulation Framework for Bitcoin-like Blockchain Technologies". In *Middleware '18: Proceedings of the 19th International Middleware Conference* (Posters), December 2018 pp. 5-6. [On line]. https://doi.org/10.1145/3284014.3284017

[8] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance benchmarking and optimizing hyperledger fabric blockchain platform". In *Proceedings of 26th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems,* MASCOTS 2018, pp. 264-276. [On line]. https://doi.org/10.1109/MASCOTS.2018.00034

[9] D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, and C. Qijun, "A review on consensus algorithm of blockchain". In *IEEE International Conference on Systems, Man, and Cybernetics, SMC 2017,* 2017-Janua, pp. 2567-2572. [On line]. https:/ /doi. org/ 10.1109/SMC.2017.8123011

[10] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System,* 2008.

[11] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling Blockchain: A Data Processing View of Blockchain Systems". In *IEEE Transactions on Knowledge and Data Engineering,* vol. 30, n.° 7, pp. 1366-1385, 2018. [On line]. https://doi.org/10.1109/TKDE.2017.2781227

[12] R. Yasaweerasinghelage, M. Staples, and I. Weber, "Predicting Latency of Blockchain-Based Systems Using Architectural Modelling and Simulation". In *Proceedings of IEEE International Conference on Software Architecture, ICSA 2017,* pp. 253-256. [On line]. https://doi.org/10.1109/ICSA.2017.22

[13] Y. Aoki, K. Otsuki, T. Kaneko, R. Banno, and K. Shudo, "SimBlock: A Blockchain Network Simulator". In *INFOCOM 2019 - IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS 2019,* pp. 325-329. [On line]. https://doi.org/10.1109/INFCOMW.2019.8845253

[14] X. Xu, Q. Lu, Y. Liu, L. Zhu, H. Yao, and A. V. Vasilakos, "Designing blockchain-based applications a case study for imported product traceability", *Future Generation Computer Systems,* vol. 92, pp. 399-406. https://doi.org/10.1016/j.future.2018.10.010

[15] H. Sukhwani, J. M. Martínez, X. Chang, K. S. Trivedi, A. Rindos, "Performance modeling of PBFT consensus process for permissioned blockchain network (hyperledger fabric)". In *Proceedings of the IEEE Symposium on Reliable Distributed Systems,* Sept. 2017, pp. 253-255. [On line]. https://doi.org/10.1109/SRDS.2017.36

[16] R. A., Memon, J. P., Li, and J. Ahmed, "Simulation model for blockchain systems using queuing theory", *Electronics (Switzerland),* vol. 8, n.° 2, pp. 1-19. [On line]. https://doi.org/10.3390/electronics8020234

[17] C. Faria and M. Correia, "BlockSim: Blockchain simulator". In *Proceedings of 2nd IEEE International Conference on Blockchain,* Blockchain 2019, pp. 439-446. [On line]. https://doi.org/10.1109/Blockchain.2019.00067

Este libro se terminó de imprimir
en los talleres de Divegráficas S.A.S
en el mes de diciembre de 2020

El libro Investigación e Innovación en Ingeniería de Software, en su cuarto volumen, compila once capítulos que relacionan las disciplinas y tópicos de la industria 4.0, con referencia a la especificación de requisitos de software, la inteligencia de negocios, la gestión del conocimiento, los sistemas de información geográfica, la seguridad de la información, la lingüística computacional y la inteligencia artificial.

Tecnológico de Antioquia
Institución Universitaria
VIGILADA MINEDUCACIÓN

SELLO EDITORIAL TdeA

GiiSTA
Grupo de Investigación en Ingeniería de Software del Tecnológico de Antioquia