BML: A Data Mapping Language for Blockchain Platforms

Ba-Lam Do, Quang-Thang Nguyen, Thang Nguyen, Tien-Thao Nguyen, Thanh-Chung Dao, BinhMinh Nguyen
School of Information and Communication Technology, Hanoi University of Science and Technology, Hanoi, Vietnam
{lamdb, chungdt, minhnb}@soict.hust.edu.vn
{thangnq, thang.nguyen, thao.nguyen}@v-chain.vn

***Abstract*—Currently, all blockchain-based applications conduct two primary operations, i.e., writing data on blockchain networks and reading these data from the networks. These tasks require users to have considerable knowledge in blockchain technology, and they become even more challenging if users want to utilize different blockchain platforms to write and read data. So far, we have not had a uniform mechanism to perform write and read operations on various blockchain platforms. In addition, writing a huge amount of data on blockchain networks is a time-consuming task and requires considerable transaction fees. To address these issues, we present in the paper a data mapping language named BML. BML allows developers to uniformly define mappings for data transformation from traditional data storage mechanisms into blockchain networks. Conversely, this language also assists users in reading transformed data. Currently, BML accepts five input data sources, including XML, JSON, XLSX, SQL (relational database), NoSQL, and supports two output platforms, including Hyperledger Sawtooth and Ethereum.**

***Index Terms*—Blockchain, data read, data write, ipfs, mapping language**

I. Introduction

In recent years, blockchain has been considered as one of the promising technologies that can make considerable changes to operations of companies, organizations, and governments over the world [1]. A large number of blockchain-based applications have been developed in multiple domains such as economy, education, transportation, health, etc. This technology is predicted to break out of impact degree over the next five years, according to the report of the top ten strategic technologies in 2020 of Gartner [2].

Although blockchain-based applications can have different characteristics of data, process, and domain, all of them make use of two primary operations: writing data on blockchain networks and reading these data from the networks. Writing data is the process of pushing data of organizations, that are stored in files such as XML, JSON, CSV, or database management systems such as SQL Server, MySQL, Elasticsearch into blockchain networks. Furthermore, additional processing steps for data cleaning and standardization can be performed before writing data permanently. Conversely, reading data allows users to retrieve data stored on the blockchain to establish undeniable proofs. In order to read and write data, developers can choose between two main approaches. On the one hand, developers can deploy a full node that has a direct and authenticated connection to a blockchain network. This node needs to have enough storage capacity and computation power to maintain a copy of all transactions as well as validate candidate transactions. By using such a node, developers can use commands supported by the blockchain platform to directly access the network [3]. On the other hand, developers can rely on third parties called API providers or blockchain endpoints to read data from and write data into blockchain networks.

However, it can be seen that there are three main issues in existing approaches. Firstly, a generic mechanism that allows developers to interact with different blockchain platforms, in the same manner, has not yet been provided. The reason for this is that each platform has a set of special commands used for accessing data, whereas existing API providers typically only focus on popular blockchain platforms such as Bitcoin [4], Ethereum [5]. The ability to interoperate between different blockchain networks will provide developers with extra options for enhancing transparency and earn trust from users [6]. Secondly, existing approaches only provide limited support for developers in data write operations. Currently, developers must define methods by themselves to select preferred data from the original data source, then transform data into a suitable representation. They only receive support from API providers and blockchain platforms for the last step to store data on the target blockchain network. Therefore, it is necessary to have a comprehensive approach to support developers in data write operations. Lastly, existing approaches do not pay significant attention to consumption of transaction fees and processing time, which users spend for operations of data read and write. These costs are considerable when users need to write a huge number of data on blockchain networks. For example, Ethereum public network can only handle at most 20 transactions per second [7], [8]. On Oct. 01, 2020, 6,425 new generated blocks on Ethereum with an average block size of 36,799 bytes paid more than 2,7 million USD in total fees for writing data (rate: 1 ETH = 346.47 USD) [9]. As a result, to write 1 KB, users need to pay approximately 11 USD. Therefore, it is necessary to compress the input dataset before writing the compressed data into blockchain networks. This mechanism needs to satisfy two important requirements, including immutable characteristics of stored data and suitable speed for writing and reading data.

To address the three challenges above, we present in this paper a data mapping language named BML. The purpose of this language is to provide a unified and effective mechanism for writing data to and reading data from blockchain platforms. Particularly, to solve the first two challenges, BML is composed of adapters that provide a query layer upon blockchain platforms together with processors that are responsive to analyzing input data sources. Based on developed adapters and processors, BML utilizes a set of generic concepts to allow users to define customized rules for data access. To address the last issue, BML makes use of InterPlanetary File system (IPFS) [10] – a peer-to-peer version-controlled system with high-throughput and decentralized distribution, for effectively storing the input data. IPFS addresses a file by a unique content identifier (CID), which is the cryptographic hash of the content of the input file. Therefore, we can utilize IPFS as a place to directly store the encrypted data while their CIDs are stored on the blockchain networks. Our experiments show that using IPFS not only reduces transaction fees but also facilitates speed of data read and write. Currently, BML supports five input data types, including XML, JSON, XLSX, SQL (relational database), NoSQL, and accepts two output platforms, including Ethereum and Sawtooth.

The remainder of this paper is organized as follows. In Section II, we discuss related work. Section III presents the design of BML and Section IV illustrates such design by means of detailed implementation. Section V discusses evaluation of BML and we conclude in Section VI with an outlook on future research.

II. Related Work

We organize the related work to this paper into three main categories, including research on mapping languages, intermediate languages used in blockchain, and data query on blockchain networks.

Regarding the former group, a significant number of mapping languages that describe rules for data transformation from one format into another format have been developed. Indeed, XLWrap [11] allows users to transform data in spreadsheets and CSV files to RDF format [12]. XSPARQL [13] is a combination of two languages, namely SPARQL and XQuery, to define data transformation between XML and RDF formats. To build this language, the authors rely on the syntax of XQuery and define some new components that are available in SPARQL syntax. D2RMap [14] is a declarative language using XML syntax to define mappings between relational databases and RDF. R2RML [15] is the only mapping language that has become a W3C recommendation for data transformation. Its mappings are written in Turtle RDF syntax to transform tables, views, or valid SQL queries of input relational databases into RDF format. RML [16] is an extension of R2RML, which provides a uniform mapping description for data transformation of an arbitrary input format into RDF. Database related concepts in R2RML such as a logical table, table name, column, etc. are replaced by new generic concepts, i.e., logical source, source name, reference, respectively.

Research on intermediate languages for blockchain platforms has received interest of researchers. Egelund-Müller et al. [17] integrated a contract language [18] which is used for expressing financial agreements, into Ethereum. The authors proposed an architecture that can separate the contract description from strategies for its execution. The use of a distributed ledger such as Ethereum will facilitate the automated execution of financial commitments. Hildenbrandt et al. [19] introduced KEVM, a formal specification for Ethereuum Virtual Machine to analyze and verify smart contracts. KEVM includes three main components, i.e., language's syntax, description of state, and control flow to execute program. Bragagnolo et al. [20] proposed a query language named EQL (Ethereum Query Language) that enables users to search for information from Ethereum through SQL-like queries. EQL provides a richer syntax and higher abstraction level to specify data, hence simplifies the searching task.

To put our work in a broader context, we contrast it to approaches of data query on blockchain networks. Galici et al. [21] introduced a system that can extract data on blockchain networks, then transform and load them into a relational database. The proposed system satisfies the requirements of scalability and usability of data analysis. Li et al. [22] presented EtherQL, an efficient query layer to support analytical queries for Ethereum. In EtherQL, data can be automatically synchronized from Ethereum network in real-time, which provides developers a convenient query layer to the whole data. Etherscan [23], BlockCypher [24], Blockchain Explorer [25], and Binance [26] are blockchain endpoints that support APIs for querying blockchain data in popular platforms such as Bitcoin [4], Ethereum [5].

III. Language Design

In this section, we first introduce main characteristics of BML in Section III-A. Next, syntax of data write and read operations will be introduced in Sections III-B and III-C, respectively.

A. Main Characteristics

BML is designed to become a generic mapping language that allows the transformation of heterogeneous sources into blockchain through users' designated rules, then query such data from the blockchain networks. To this end, we design BML with six salient characteristics, including uniformity, efficiency, convenience, inheritance, openness, and linkage.

Uniformity: Currently, data of organizations can exist in multiple representations such as JSON, XML, SQL, NoSQL, etc., which have their own characteristics. For example, JSON stores data in objects based on key-value pairs, whereas XLSX operates data on worksheet tables. SQL (relational database) manages data in tuples, relations, and schemata, while NoSQL needs no schema to tie data. BML aims to write data from an arbitrary input format into blockchain networks through mapping rules. In addition, mappings should retain similar syntax even when used with different formats, which facilitates easy-to-use. To this end, BML relies on a set of processors for analyzing input data and adapters for accessing blockchain networks. Upon these components, BML provides a collection of generic concepts and utilizes the same structure to describe data transformation for different inputs.

Efficiency: Compared to current database management systems, blockchain platforms have limited transaction processing performance [27], [28]. In addition, public blockchain networks such as Bitcoin and Ethereum require users to pay significant fees for mining nodes to add new data into the networks. Applying scalability techniques has not created remarkable improvements because this aim requires to redesign fundamentals of blockchain technology such as constraints of block validation, avoid double spending [29]. To overcome these issues, BML relies on IPFS to store input data and conduct data read and write operations in parallel. On the one hand, BML needs only one transaction to store information (i.e., CID) of the input data into blockchain networks, which makes BML independent of the magnitude of input data. On the other hand, BML allows users to significantly reduce the processing time.

Convenience: We select JSON syntax for describing requirements of users, including both data transformation and query, because this format is an ISO standard used for structured data interchange between organizations and individuals. JSON format is not only comfortable for users to read and write but also simple to parse by programming languages. Furthermore, JSON is a lightweight format; hence, the sizes of mappings and queries are small, which is highly convenient for storage.

Inheritance: Although the mapping definition is a quite straightforward task, an emerging issue is that there will be repetitive tasks for users when they need to handle similarly structured input data. To address this issue, BML allows users to reuse definitions from previously defined mappings. This characteristic helps users reduce the time and effort needed for defining data transformation mappings with a similar structure.

Openness: A high demand in data transformation and query is to preprocess input data and filter query results. BML has this capability by allowing users to declare predefined functions using the syntax of Python programming language when describing mappings and filters. This characteristic not only releases BML from entirely textual descriptions but also opens the adaptability to different scenarios for BML.

Data linkage: An important feature of BML is the ability to link data between related sources in data transformation. In particular, users can define mappings to integrate data from multiple sources before storing integrated data on blockchain networks. As a result, they can query from a single data point on the blockchain network, instead of querying from multiple data points, which leverages efficient data querying.

B. Data Write

To illustrate the useful characteristics of BML for writing data, we utilize four examples, including one input JSON file and three mappings for JSON, XML, and SQL data sources. The extracts of these examples are represented in Listings 1, 2, 3, and 4, respectively whereas the full descriptions are available at https://gitlab.com/bkc-lab/bml/-/tree/master/mappings. In each mapping, object *mapping* is an array of definitions, that allows users to describe and link different mappings; each consists of four parts: source, identifier, reference, and function.

- **source**: this part contains three key-value pairs, which allows BML's processor to determine and gather input data. The first attribute, i.e., *path* identifies the location of the data source where its value can be a path if the input is a flat-file such as JSON, XML, XLXS, etc. or a connection string in the form of *host:port/username/password/database name* if the input is a database. The next field, *type*, indicates the input format, while the final attribute, *iterator*, determines the iteration pattern on the input source to extract the data that users are interested in. For example, the pair of *iterator: profile* in Listing 2 indicates the iteration over *profile* object on the JSON file. With relational databases, the value of *iterator* represents the selected table in the database, whereas, with non-relational databases such as Elasticsearch, this value can be an index.
- **identifier**: allows users to assign an identifier to each mapping. Identifiers of different mappings defined by the same user should be differentiated. This component will be used in three cases: (i) data read: users need to declare the *identifier* of the mapping to identify the data they want to query on blockchain network; (ii) data linkage: allows users to link different data sets following the syntax of *mappingID(attribute1, attribute2)*. In which *attribute1* and *attribute2* are used to link the current mapping and the mapping identified by *mappingID*. For example, Listing 2 contains *SQLMapping(id, user_id)*. As a result, field *user_id* in the mapping *SQLMapping* of Listing 4 is linked to filed *id* in the mapping *JSONMapping* of Listing 2; (iii) mapping reusing: for example, the *XMLMapping* mapping defined in Listing 3 is reused in Listing 2 to perform the same transformation for three attributes including name, phone, and email.
- **reference**: this component is an array containing mapping definitions to transform input attributes to output attributes. Each mapping definition is written in the syntax of *attribute_src→attribute_des* in which *attribute_src* is the attribute from the input data source. Then, its values will be saved as values of *attribute_des* on the blockchain network.
- **function**: this component contains an array of definitions, and each identifies a function that manipulates input data. BML utilizes Python-based syntax to declare functions, i.e., *attribute→function(type(attribute))*. For example, *user_name → str(user_name).upper()* in Listing 2 will transform values of attribute *user_name* to uppercase letters.
- **blockchain**: this object refers to the target blockchain network for data write and read operations where the interaction between users and the blockchain is carried. Detailed information of each network (type, blockchain

endpoint's address, etc.) are given by users when they create their accounts.

Listing 1: An extract of JSON file

```
{
  "profile": [
      {
          "id": "",
          "name": "",
          "age": "",
          "address": {
              "latitude": "",
              "longitude": "",
          },
          "job": "",
          "phone": "",
          "email":""
      }
      ...
  ]
}
```

Listing 2: A sample mapping for JSON format

```
{
  "mapping":[
      {
          "source":{
              "path":"http://localhost/data.json",
              "type":"JSON",
              "iterator":"profile"
          },
          "identifier":"JSONMapping",
          "reference":[
              "id->id",
              "age->date_of_birth",
              "address.latitude->address.latitude",
              "address.longitude->address.longitude",
              "SQLMapping(id, user_id)",
              "XMLMapping()"
          ],
          "function":[
              "user_name->str(user_name).upper()",
              "date_of_birth->2020-int(date_of_birth)"
          ],
          "blockchain":"Ethereum"
      }
  ]
}
```

Listing 3: A sample mapping for XML format (excerpt)

```
{
  "identifier": "XMLMapping",
  "reference": [
      "name->user_name",
      "phone->phone",
      "email->email"
  ]
}
```

Listing 4: A sample mapping for SQL data source (excerpt)

```
{
  "identifier": "SQLMapping",
  "reference": [
      "id->user_id",
      "number->number",
      "date->date"
  ]
}
```

C. Data Read

Listing 5 introduces descriptions in a sample query that contains two main components.

- **identifier**: indicates the corresponding mapping used to transform data into the blockchain network. Using this mechanism, BML liberates users from remembering the exact address of block where stores the CID of the input data.
- **filter**: is an array of conditions to refine query results which utilizes Python-based syntax as follows: *attribute→condition(type(attribute))* in which *attribute* is the attribute of the mapping *indentifier*, and *condition(type(attribute))* is the rule to filter out unwanted values.

Listing 5: An extract of query description

```
{
  "identifier":"JSONMapping",
  "filter":[
      "id->int(id)>300",
      "data_of_birth->int(data_of_birth)>1995",
      "email->soict.hust.edu.vn in str(email)"
  ]
}
```

IV. Implementation

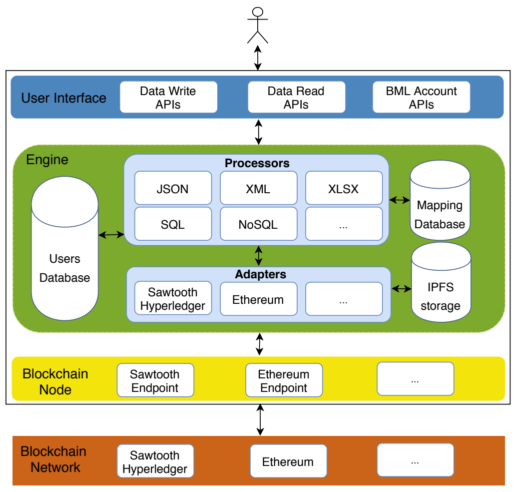


Fig. 1: System architecture

Figure 1 introduces the architecture of BML system. The system consists of three main layers, namely user interface, engine, and blockchain node.

User interface: provides a high-level interaction layer to users, which alleviates any technical difficulty in data transformation, query, and authentication. Users can either define a description file, e.g., JSONMapping.bml, then send it to the engine through the syntax of *mapping=JSONMapping.bml* or embed these descriptions in the body of an API-based request. After completing data transformation, the interface will return a guide for querying data to the user (as Swagger documentation).

Engine: is the central processing layer in BML language, which includes a set of processors, adapters, and databases. We rely on the number of CPU cores to perform processing operations in parallel. Particularly, based on mapping definitions declared by the user, a corresponding processor will be called to gather data. The processor first splits the input data into n smaller segments in which n is the number of CPU cores. In the next step, data are transformed into a common representation (i.e., JSON format), which allows us to minimize the effort of developing adapters for storing different data formats on blockchain platforms. After that, the processor encrypts data parts according to Advanced Encryption Standard (AES) before sending encrypted data to a blockchain adapter to store these parts into a folder on IPFS network. At the end of parallelization, the adapter combines the CID of the folder with information such as the user's private key to generate transaction content and delivers it to the "Blockchain node" layer. For read operations, the system queries data from the blockchain network to receive the CID, then uses it to locate parts of data stored on IPFS network, parallelizes the decoding and filtering operations before aggregating results, and returning them to the user.

Blockchain node: contains a set of our full nodes and public blockchain endpoints. These nodes can direct access to blockchain networks; hence, they are used to write and query data.

V. Evaluation

The purpose of this section is to demonstrate the ability of BML in aspects of writing data to and reading data from different blockchain networks. Moreover, we illustrate the performance and scalability of BML by the use of multi-process parallelization and different input data sizes. Note that there is no baseline comparison in our experiments since a similar approach processing JSON, XML, XLSX, SQL, and NoSQL does not exist to the best of our knowledge. We also discuss our view of privacy and decentralization of BML.

A. Experiment Information

We conduct three experiments on two blockchain networks, including (i) Hyperledger SawtoothFourth [30]: a private network including five nodes and using PBFT consensus algorithm; (ii) Ethereum [5]: the test net network using Proof-of-work consensus algorithm, available at https://ropsten.etherscan.io.

- Experiment of performance: We measure the consuming time for data write and read operations on different input data types, including XML, JSON, XLSX, SQL (we use MySQL version 5.7), and NoSQL (we select Elasticserach version 7.2). In this experiment, every input data set has similar content with 1,000,000 records, each containing nine attributes. We perform the experiment on a computer having 64-core CPU and 128 GB RAM (named machine 1).
- Experiment of scalability: We conduct the experiment with the change on the magnitude of record numbers from 1,000 to 4,000,000 for JSON format. This experiment is performed on machine 1.
- Experiment of parallelization: We repeat the second experiment on two different computers, including machine 1 with 64-core CPU and 128 GB RAM; machine 2 with 4-core CPU and 12 GB RAM. The aim of this experiment is to compare the difference in processing time on different machines, thereby allowing us to evaluate the benefit of multi-process parallelization. In this experiment, we select Hyperledger Sawtooth as the target blockchain to manage data.

We conduct each experiment ten times and calculate the average values in each test case. Note that in all our experiments, data write refers to the operation of storing input data on IPFS network and a target blockchain platform, whereas data read aims to find a unique record.

B. Experiment of Performance

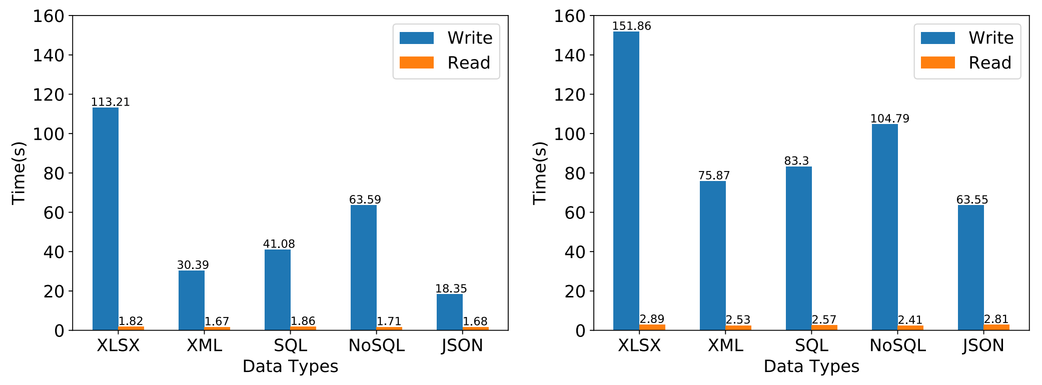
Figures 2a and 2b show write and read time on Hyperledger Sawtooth and Ethereum, respectively. These figures show that for five input data types, the data write operation for 1,000,000 records only takes less than three minutes while the read time is less than three seconds. The use of IPFS and parallelization mechanism, therefore, shows an efficient performance of BML compared to storing all input data on blockchain networks. In addition, while the peak of write time on Hyperledger Sawtooth is 113.21 seconds, it is 151.86 seconds on Ethereum blockchain. The reason is that Sawtooth is a private blockchain, including only five nodes, whereas Ethereum is a global public blockchain having higher latency for confirming and reading block.

C. Experiment of Scalability

Figures 3a and 3b show execution time on Hyperledger Sawtooth and Ethereum platforms when the input data in JSON format increases from 1,000 to 4,000,000 records. For the horizontal axis, which represents the number of records, we use the fourth root function to calculate the position of the points on this axis. The use of this function allows us to better represent the variation in time in a wide range of values. These figures depict that the increase in execution time is significantly smaller than the proliferation of data size. This experiment, therefore, shows the trusted scalability of BML with different sizes of data.

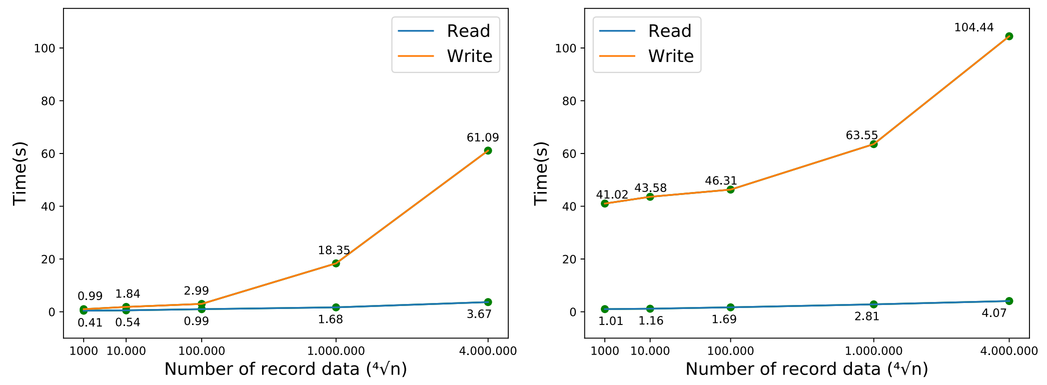
D. Experiment of Parallelization

Figures 4a and 4b describes the difference of time execution for reading and writing data on two machines (the machine 1 has 64-core CPU, 128 GB RAM while the machine 2 has 4-core CPU and 12 GB RAM). For small data (less than 100,000 records), two machines have an inconsiderable discrepancy in both operations of reading and writing data, which can be ignored. However, for the larger input data, the efficiency of the first machine is clearly revealed when reading and writing times reduce from 8.5 to 9.0 times compared to the second machine. On the one hand, this scenario shows that parallelization depending on the configuration of the input device. On the other hand, it confirms the effectiveness of utilizing the computing resources on each machine.



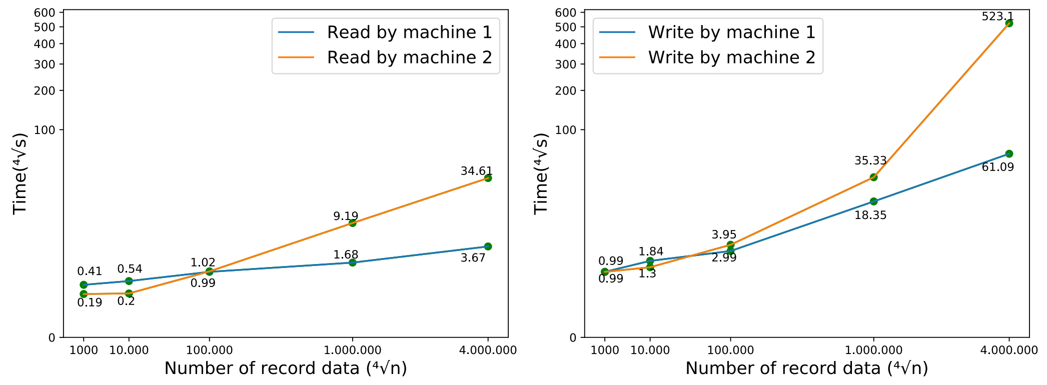
(a) On Hyperledger Sawtooth (b) On Ethereum

Fig. 2: BML Performance



(a) On Hyperledger Sawtooth (b) On Ethereum

Fig. 3: BML Scalability



(a) Read time (b) Write time

Fig. 4: BML Parallelization

E. Privacy and Decentralization

Privacy: Ensuring that unauthorized personnel cannot write data to and read data from users' chains is a significant concern. Currently, we address this issue by combining user authentication and cryptography mechanism. Firstly, all data read and write operations require the user to enter authentication codes to confirm desired actions. Next, the input data is encrypted according to the AES standard to ensure the confidentiality of stored information. Each user's private key is also encrypted according to this standard with the key as the user password; hence, the users' database will be protected from sabotage or data breach, even if attackers obtain it.

Decentralization: A salient characteristic of blockchain technology is the decentralized nature, which does not depend on a single third party. Currently, BML requires direct access to the users' data sources in order to transform data into blockchain networks. In that manner, BML needs to be trusted by the users in terms of ensuring integrity between users' data and stored data or not leaking data in the transformation process. This downside will be minimized through publishing the source code of BML with MIT license, which is available on GitLab at https://gitlab.com/bkc-lab/bml. In addition, the whole system can be deployed on users' machines and does not require a central server; hence, users can manage all their data.

VI. Conclusion

In this paper, we introduce a mapping language named BML for data management on blockchain platforms. BML is designed with six salient characteristics, including uniformity, efficiency, convenience, inheritance, openness, and data linkage, which makes BML a unified and efficient mechanism to write arbitrary data to and query transformed data from different blockchain networks. Currently, BML supports five popular input data types, including XML, JSON, XLSX, SQL, NoSQL, and links to two blockchain platforms, including Hyperledger Sawtooth (a private blockchain), and Ethereum (a public blockchain). Our evaluations show that BML has salient advantages of performance, scalability, parallelization, privacy, and decentralization.

In our future work, we will extend the current capabilities of BML to provide more convenient and efficient mechanisms for users. It will be necessary to improve processors and adapters in BML to allow users to transform new data types into multiple blockchain platforms. Furthermore, we will continue to enhance the performance and scalability to facilitate the use of this language. Finally, we aim to conduct a user survey to receive objective evaluations of advantages and disadvantages of BML.

Acknowledgement

This work was supported by the Vingroup Innovation Foundation (VINIF), project code VINIF.2019.DA07

References

[1] Z. Merkaš, D. Perkov, and V. Bonin, "The significance of blockchain technology in digital transformation of logistics and transportation," *International Journal of E-Services and Mobile Applications (IJESMA)*, vol. 12, no. 1, pp. 1–20, 2020.

[2] Gartner, "Gartner identifies the top 10 strategic technology trends for 2020," Available at https://www.gartner.com/en/newsroom/press-releases/2019-10-21-gartner-identifies-the-top-10-strategic-technology-trends-for-2020 (accessed 01 Oct. 2020), 2019.

[3] A. M. Antonopoulos, *Mastering Bitcoin: unlocking digital cryptocurrencies*. " O'Reilly Media, Inc.", 2014.

[4] S. Nakamoto, "Bitcoin: A peerreto-peer electronic cash system," Available at https://bitcoin.org/bitcoin.pdf (accessed 01 Oct. 2020), 2008.

[5] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[6] T. Hegnauer, "Design and development of a blockchain interoperability api," Ph.D. dissertation, Master Thesis, CSG@ IfI, University of Zürich, Switzerland, 2019.

[7] H. S. Galal, M. ElSheikh, and A. M. Youssef, "An efficient micropayment channel on ethereum," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2019, pp. 211–218.

[8] M. Schäffer, M. Di Angelo, and G. Salzer, "Performance and scalability of private ethereum blockchains," in *International Conference on Business Process Management*. Springer, 2019, pp. 103–118.

[9] Etherscan, "Ethereum average block size chart," Available at https://etherscan.io/chart/blocksize (accessed 02 Oct. 2020).

[10] J. Benet, "IPFS - Content Addressed, Versioned, P2p File System," Available: https://www.hirego.io/ (accessed 01 Oct. 2020).

[11] A. Langegger and W. Wöß, "Xlwrap–querying and integrating arbitrary spreadsheets with sparql," in *International Semantic Web Conference*, 2009, pp. 359–374.

[12] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific american*, vol. 284, no. 5, pp. 34–43, 2001.

[13] S. Bischof, S. Decker, T. Krennwallner, N. Lopes, and A. Polleres, "Mapping between rdf and xml with xsparql," *Journal on Data Semantics*, vol. 1, no. 3, pp. 147–185, 2012.

[14] C. Bizer, "D2r map-a database to rdf mapping language," in *WWW (Posters)*, 2003.

[15] S. Das, S. Sundara, and R. Cygania, "R2rml: Rdb to rdf mapping language," 2012.

[16] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle, "Rml: a generic language for integrated rdf mappings of heterogeneous data," in *Workshop on Linked Data on the Web*, 2014.

[17] B. Egelund-Müller, M. Elsman, F. Henglein, and O. Ross, "Automated execution of financial contracts on blockchains," *Business & Information Systems Engineering*, vol. 59, no. 6, pp. 457–467, 2017.

[18] P. Bahr, J. Berthold, and M. Elsman, "Certified symbolic management of financial multi-party contracts," *ACM SIGPLAN Notices*, vol. 50, no. 9, pp. 315–327, 2015.

[19] E. Hildenbrandt, M. Saxena, N. Rodrigues, X. Zhu, P. Daian, D. Guth, B. Moore, D. Park, Y. Zhang, A. Stefanescu *et al.*, "Kevm: A complete formal semantics of the ethereum virtual machine," in *IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 2018, pp. 204–217.

[20] S. Bragagnolo, H. Rocha, M. Denker, and S. Ducasse, "Ethereum query language," in *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, 2018, pp. 1–8.

[21] R. Galici, L. Ordile, M. Marchesi, A. Pinna, and R. Tonelli, "Applying the etl process to blockchain data. prospect and findings," *Information*, vol. 11, no. 4, p. 204, 2020.

[22] Y. Li, K. Zheng, Y. Yan, Q. Liu, and X. Zhou, "EtherQL: A Query Layer for Blockchain System," in *Database Systems for Advanced Applications*, S. Candan, L. Chen, T. B. Pedersen, L. Chang, and W. Hua, Eds. Springer, 2017, vol. 10178, pp. 556–567.

[23] Etherscan, "Etherscan apis," Available at https://etherscan.io/apis (accessed 01 Oct. 2020).

[24] BlockCypher, "Introduction," Available at https://www.blockcypher.com/dev/bitcoin/\#introduction (accessed 01 Oct. 2020).

[25] Blockchain, "Blockchain explorer," Available at https://www.blockchain.com/api (accessed 01 Oct. 2020).

[26] Binance, "Binance apis," Available at https://docs.binance.org/api-reference/dex-api/paths.html (accessed 01 Oct. 2020).

[27] M. Muzammal, Q. Qu, and B. Nasrulin, "Renovating blockchain with distributed databases: An open source system," *Future generation computer systems*, vol. 90, pp. 105–117, 2019.

[28] G. Chung, L. Desrosiers, M. Gupta, A. Sutton, K. Venkatadri, O. Wong, and G. Zugic, "Performance tuning and scaling enterprise blockchain applications," *arXiv preprint arXiv:1912.11456*, 2019.

[29] S. Bano, M. Al-Bassam, and G. Danezis, "The road to scalable blockchain designs," *USENIX; login: magazine*, vol. 42, no. 4, pp. 31–36, 2017.

[30] K. Olson, M. Bowman, J. Mitchell, S. Amundson, D. Middleton, and C. Montgomery, "Sawtooth: An introduction," Available at https://www.hyperledger.org/wp-content/uploads/2018/01/Hyperledger_Sawtooth_WhitePaper.pdf (accessed 01 Oct. 2020), 2018.