Big Data Architecture for Cryptocurrency Real-time Data Processing

Nebojša Horvat, Vladimir Ivković, Nikola Todorović, Vladimir Ivančević, Dušan Gajić, Ivan Luković
Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia
{horva.n, vladimir.ivkovic, nikola.todorovic, dragoman, dusan.gajic, ivan}@uns.ac.rs

***Abstract*— The role of cryptocurrencies in the alternative world economy has been steadily increasing since they emerged in 2009. However, understanding their actual influence and performance can be a difficult task. In this paper, we present an architecture for real-time cryptocurrency data processing and analysis based on the Lambda architectural approach. The proposed architecture offers both batch and stream data processing of cryptocurrency transactions and blocks, as well as analyzing various sorts of trends in a blockchain network and an exchange market. The proposed architecture is modular and thus easies the implementation of loosely coupled components and also provides several benefits for cryptocurrency analysis: real-time monitoring of blockchain events and mining statistics, cryptocurrency buying and selling trends, as well as social media events related to cryptocurrency reputation.**

I. INTRODUCTION

During the last decade, "big data" has become one of the most overused terms, both in industry and academia. This term is used to describe a wide range of concepts: from the technological ability to store, aggregate, and process data, to the cultural shift that is pervasively invading business and society, both drowning in information overload [6]. However, big data is usually defined as a large amount of data that requires new technologies and architectures so that it becomes possible to extract value from it by capturing and analyzing processes of interest [7]. Big data informally refers to data volumes in ranges that exceed the capacity of current online storage systems and processing systems. The volume is the first of the widely recognized big data properties often mentioned as "4V", others are velocity, variety, and veracity [8]. Big data originally meant the data that could not be processed (efficiently) by traditional database methods and tools [9]. The current growth rate in the amount of data collected is staggering. A major challenge for IT researchers and practitioners is that this growth rate is fast exceeding our ability to both: (1) design appropriate systems to handle the data effectively and (2) analyze it to extract relevant meaning and information. The digital age and technological progression over the globe enable the collection, analyses, and implementations of big data systems, which have been embedded into every aspect of everyday life and are progressing rapidly [10].

Starting in 2008, with Satoshi Nakamoto's Bitcoin white-paper [1] and the start of Bitcoin's peer-to-peer network, cryptocurrencies have emerged as both technological and economic phenomena. Since then, cryptocurrencies have been a trending topic, especially over the past few years, pooling tremendous technological power and attracting investments valued over trillions of dollars on a global scale. The mentioned characteristics of cryptocurrencies make data gathered from their transactions very valuable and worthy of analysis. A cryptocurrency can be defined as an encrypted digital currency that incorporates cryptography and in most cases utilizes blockchain technology. Blockchain could be regarded as a public distributed ledger and all committed transactions are stored as a list of blocks. This chain grows as new blocks are appended to it continuously. Asymmetric cryptography and distributed consensus algorithms have been implemented to ensure user security and ledger consistency. Blockchain technology generally has key characteristics of decentralization, persistence, anonymity, and auditability [11] that makes it an obvious choice for a distributed software system behind cryptocurrencies.

Every second, thousands of transactions are submitted to various blockchain networks supporting the operation of different cryptocurrencies. Those transactions are broadcasted to millions of network nodes all over the world. Most popular and also most valued cryptocurrencies, like Bitcoin and Ethereum, use Proof of Work (PoW) as their consensus algorithm [1]. In PoW, miners (nodes which maintain network) place pending transactions into blocks. Next, they try to solve the cryptographic puzzle to find a new block that can be appended to the blockchain. During the process of finding a new block, usually called mining, a large amount of data is generated and transported through the network. As cryptocurrencies are the most popular form of the alternative global economy, there exists a great need to process massive amounts of data that are gathered every day. By processing this data, we would gain a much better insight into this new growing market. Such a piece of knowledge can be used by cryptocurrency analysts for better understanding and tracking of events that happen on the network. Moreover, data processing results can produce a potential financial gain by predicting price fluctuations of cryptocurrencies.

Benefits of understanding and predicting fluctuations that happen in the cryptocurrency world motivate us to find the architecture which is best suited for processing such large quantities of data. Such a solution would serve as a real-world example on the big data Architectures course that is as a part of the master study program in Computing and Control Engineering at the Faculty of Technical Sciences in Novi Sad. This solution would also help students to fully understand the mixture of 4 Vs in big data systems.

Due to the volatile nature of cryptocurrencies, analysts, traders, and practitioners want to monitor and analyze various

types of events related to the cryptocurrency market. That requires the existence of a unified real-time analytic tool that can be used by all stakeholders in the cryptocurrency world.

In order to find optimal architecture for such a tool, we need to think of a way to manage continuous data streams that should be processed in real-time. Furthermore, all transactions and blocks previously created and combined with a history of price changes and social media events need to be stored for possible future processing. The problem of storing such versatile and extensive data must be solved in a scalable and distributed way. The way of storing data is closely combined with the way it is processed. Thus, we must also find tools and frameworks that provide a scalable data processing algorithm that can in a fast way access accumulated data in order to produce results in a short time-frame.

II. RELATED WORK

Many researchers have worked on the problem of processing cryptocurrency data and combining it with social media data to make predictions on cryptocurrency prices or find some other patterns. In [2] and [3], various social media posts and their sentiment were used to find correlations with cryptocurrency trading and cryptocurrency price history. Further, in [4], a platform was created for real-time cryptocurrency price prediction. They also studied the relationship between news sentiments and bitcoin price fluctuations and developed a prediction model for the price in the next minute.

In [5] model-less convolutional neural networks are used with the historic price of a set of financial assets as its input, outputting portfolio weights of the set. The network is trained with 0.7 years' price data from a cryptocurrency exchange. The training period of a network is 30 minutes. In order to use this model in real online trading, it is needed to put the training set at a closer time to the current time, or even do online training while trading. Such models can be used for predictions using historical data but lack a convenient and efficient way to process new data in real-time.

Although the literature is extensive, we could not find related papers that utilize Lambda architectural approach that would combine batch processing (which uses historical data) and streaming processing (which uses recently generated data) to create a scalable hybrid architecture.

III. LAMBDA ARCHITECTURAL APPROACH

Designing an architecture for processing and analyzing cryptocurrency data can result in a structurally complex system and may include many different technologies. If we want to take into account all transactions, blocks, and other trading and social media historical data, the system's complexity additionally increases. While relying on historical data, the system needs to retain the ability to also process real-time data and combine it with previously processed data to create the whole picture. In order to solve this problem, we propose the use of the Lambda architectural approach. The Lambda architecture, first proposed by Nathan Marz [12], provides a way to implement an arbitrary function on arbitrary data in real-time. It is a high-level approach for big data processing, and it is considered one of the industry's best practices for scalable real-time big data processing. While working with very large data sets, it can be time-consuming to extract information through required queries. These queries cannot be executed in real-time, and often require algorithms and methods such as MapReduce that can process data in parallel across the entire data set. The results are later stored separately from the original data and used for querying. One drawback of this approach is that it can introduce latency. If the processing takes a few hours, a query may return results that are several hours old. Analysts usually prefer getting the latest results in real-time to combine these results with the results from the previous batch analysis. The Lambda architecture addresses this problem by creating two paths for data flow – cold path and hot path. All data coming into the system go through these three layers:

- a batch layer (cold path) stores all of the incoming data in its raw form and performs batch processing on the data; afterward, the result of this processing is stored as a batch view,
- a serving layer (cold path), responsible for indexing the batch views for efficient querying; it is also being updated by the speed layer with incremental updates based on the most recent data, and
- a speed layer (hot path) analyzes data in real-time providing real-time (speed) views, so this layer is designed for low latency, at the expense of accuracy.

IV. PROPOSED BIG DATA ARCHITECTURE

Our proposed solution architecture is composed of several components, and is an adaptation of the Lambda architectural approach, in the cryptocurrency domain. Graphical representation of our Lambda architecture adaptation with auxiliary components is shown in Fig. 1. The figure consists of: (1) data source represented as a set of different data streams, (2) message bus responsible for communication between components, (3) three processing layers including batch layer, speed layer, and serving layer, and (4) client application that has an appropriate user interface. The following subparagraphs contain a more detailed description of the data source and processing layers.

A. Data Sources

The data source consists of two major data streams: (1) recently submitted (pending) transactions and (2) new blocks appended to the blockchain. An additional stream can be real-time crypto exchange data (prices change every second), social network mentions of cryptocurrencies, or something not directly related to the cryptocurrency such as global stock market data or oil price changes. Apache Kafka is utilized as a messaging platform because it enables distributed, fault-tolerant, and fast messaging using parallelization and partitioning techniques. There are several possible ways to obtain blockchain data. The easiest and quickest way would be using public APIs to retrieve new blockchain data, but this approach could cause certain difficulties regarding a limited

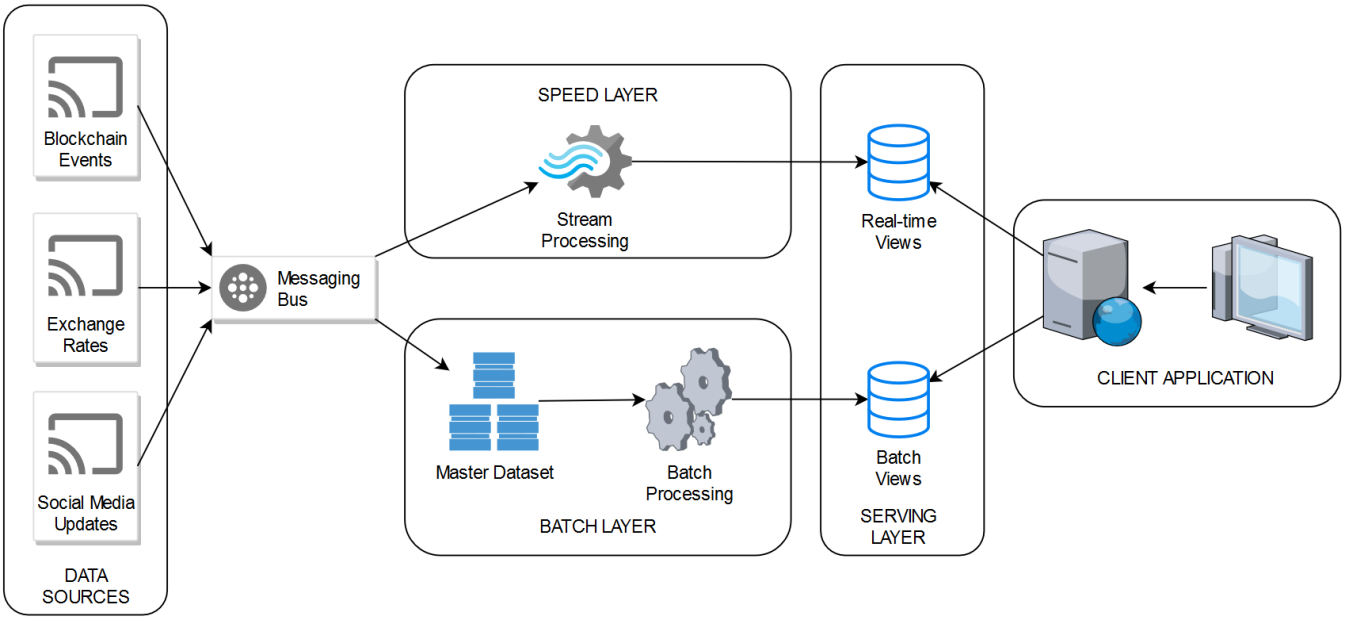

Fig. 1. The graphical representation of the proposed big data architecture

number of API calls, establishing a permanent connection to public APIs, as well as obtaining data in real-time. Another way is to host one light-weight node for each cryptocurrency of interest. A light-weight node usually stores the earliest part of the blockchain and it is synchronized with the nearest blockchain network nodes. Most of the cryptocurrencies Software Development Kits enable webhooks or publish-subscribe mechanisms to track events on the blockchain network in real-time. Such events are then sent to appropriate Kafka topics and can be consumed by any other component in the system. Exchange rates data includes historical and latest cryptocurrency prices in major fiat currencies, e.g. USD or EUR. There are numerous free APIs offering the latest prices of cryptocurrencies at particular crypto exchanges. Most of these services require registration, after that the API key is provided, and every API call should be performed with that API key. Some of the popular cryptocurrency price APIs are CoinAPI[1], CoinMarketCap[2], and CryptoCompare[3]. Social media data can be anything collected from news websites, popular blogs, and social and professional networks that is relevant and related to certain cryptocurrency or the cryptocurrency world in general. At least, we should consider popular posts from social networks, e.g. Twitter[4] and Reddit[5]. Both of these networks have public APIs for real-time data acquisition, so we can retrieve cryptocurrency-related posts to have information on the latest news and events. Twitter API offers real-time filtering of tweets by keywords and hashtags, as well as the latest trending topics for specified geolocation. Reddit API enables real-time retrieval of posts and comments associated with specified subreddit. Another important source of the latest cryptocurrency news can be popular blockchain websites such as CoinDesk[6], CoinTelegraph[7], and Crypto Invest[8]. Content from these and similar websites could be used to assess the community's attitude about particular cryptocurrency topics.

B. Batch Layer

The Hadoop Distributed File System (HDFS) and Apache Spark are the technologies used for the batch layer of the proposed architecture. HDFS is chosen as a storage for the master dataset because it is distributed, scalable, portable, and fault-tolerant. Distribution of the HDFS is achieved using Docker containers. Spark SQL, as a module of Apache Spark, is utilized to implement batch view computations. Spark SQL represents a high-level tool commonly used for processing structured datasets.

The Master dataset consists of the following major entity types: blocks, transactions, and social media posts. Block data contains basic block info such as block hash, timestamp, previous block hash, block height in blockchain, total transaction volume, and a number of transactions. Additionally, block data can include mining details: block difficulty, Merkle root, and block reward for the miner. Transaction data includes general attributes: transaction hash, transferred quantity of cryptocurrency, fee, timestamp, and size in bytes. Furthermore, for different cryptocurrencies, we could store specific attributes for block mining or transaction process, e.g. gas price and the gas used for Ethereum. Social media posts contain post content in free text form, source, timestamp, one or more cryptocurrencies the post is related

[1] www.coinapi.io
[2] www.coinmarketcap.com/api
[3] http://min-api.cryptocompare.com
[4] http://developer.twitter.com/en/docs
[5] www.reddit.com/dev/api
[6] www.coindesk.com
[7] www.cointelegraph.com
[8] www.cryptoinvest.tech

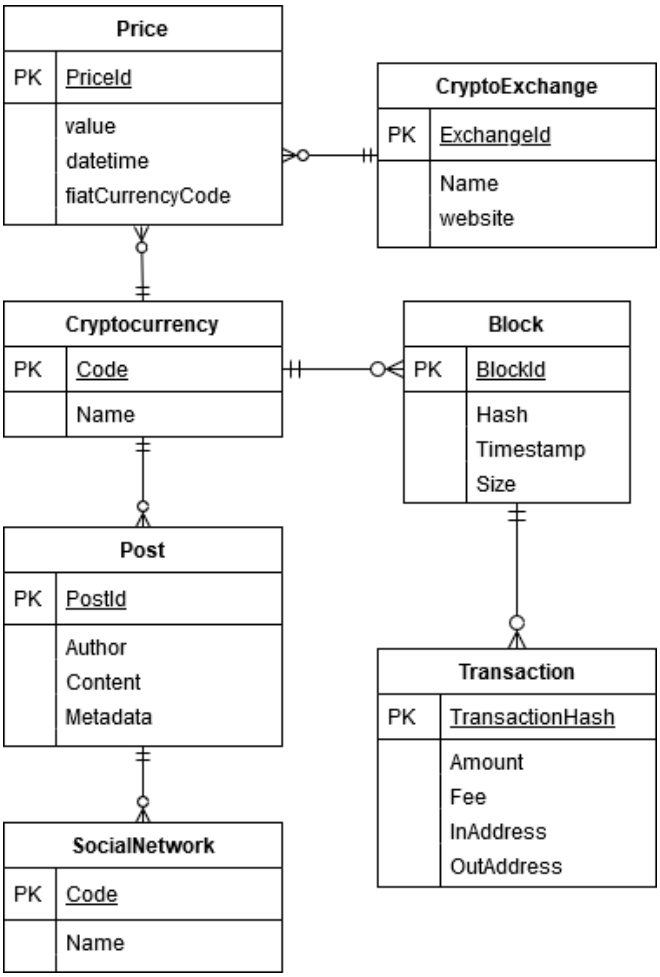

Fig. 2. A conceptual entity-relationship diagram of master dataset

to. Master dataset is formed by appending new data records consumed from several Kafka topics, one for each entity type. HDFS stores data divided into separate directories for each entity type containing multiple text files in JSON format. Conceptual entity-relationship diagram is shown in Fig. 2.

Batch processing procedures include Spark SQL jobs that are performed on data stored in HDFS. Spark SQL enables loading data from JSON format into Spark DataFrames. Every few hours, a simple bash script or some dedicated job-scheduling tool initiates Spark SQL jobs batch processing. The results of these jobs are sent to appropriate Kafka topics and eventually stored in the serving layer.

An example of such a job would be calculating the correlation between the price of cryptocurrency and its popularity on social networks. Master dataset is saved on HDFS so we first need to read data from directories that contain the price of a cryptocurrency. Data is loaded into Spark as a data frame that contains a timestamp and the corresponding price. Afterward, social media data about the popularity of cryptocurrency needs to be read and loaded into the Spark context resulting in multiple data frames (each for the different social networks). In order to simplify our cryptocurrency popularity metric, all activity on social networks that happened in a given time window was counted. That includes the latest posts associated with certain cryptocurrency and all its reactions and comments. After combining all social media data we should get a data frame that contains the "average" popularity of cryptocurrency in a certain time interval throughout different social networks and websites. Having data prepared in such a way makes it possible to calculate any type of correlation between cryptocurrency popularity and price during the last hours or days.

C. Speed Layer

Real-time data processing in the Speed layer can be implemented utilizing another Apache Spark module called Spark Streaming or Kafka Streams API. Spark Streaming is an extension of core Spark API that enables scalable, high throughput processing capability on streaming data. Spark Streaming divides data into micro-batches, which are processed further to generate a stream of results in batches. Kafka Streams is a client library for building data stream processing applications, where the input and output data are stored in Kafka clusters.

Stream processing jobs include filtering, mapping, aggregation, and join procedures performed on data streams of interest. An example of a stream processing job refers to the same correlation calculation process described previously in the Batch layer chapter. The high-level procedure is very similar, but the main difference is in the computing paradigm itself. Stream processing procedure takes into account only real-time data as an input. Suppose Kafka Stream API is chosen as a stream processing library. The required input KStreams will be formed by consuming records from Kafka topics with cryptocurrency prices and the latest social media activities. Data aggregation of input streams will result in KTables that can be joined enabling a calculation of correlation between cryptocurrency price changes and its popularity on social and professional networks.

D. Serving Layer

This architecture proposal utilizes Apache Druid as a serving layer database to store data records required from the batch and real-time views. Druid is an open-source distributed column-oriented database management system that combines ideas from analytic databases, time-series databases, and search systems to enable use-cases in streaming architectures. It provides fast analytical queries, at high concurrency, on event-driven data, and ad-hoc analytics on both real-time and historical data. Druid integrates natively with message buses such as Kafka.

Druid is capable of both real-time and batch ingestion, so it is the perfect choice for the serving layer. The general idea behind Druid's real-time ingestion setup is that events are sent, as they occur, to a message bus, in our case Kafka, and then Druid's real-time indexing service connects to the bus and stores the data as it comes. On the other hand, Druids natively supports Apache Hadoop-based batch ingestion via a Hadoop-ingestion task. For the Hadoop-ingestion task, it

is necessary to specify the data source, data format (JSON in our case), and attributes of interest called dimensions in Druid. The data source includes HDFS location containing results of the previously described Spark batch jobs. Druid's data model supports both the relational and time-series nature of stored data. Druid ingestion schemas must always include a primary timestamp that is further used for partitioning and sorting your data. The timestamp attribute is one of the major ones in the world of blockchain. Every transaction, block, or exchange rate has a corresponding timestamp. Therefore, Druid's primary timestamp perfectly suits the blockchain data model.

Choosing the Apache Druid database for storing both batch and speed views eases the implementation of the Serving layer. Therefore, it is not required to create and maintain additional merge-view procedures as it was necessary if we were using separate databases for batch and speed views. An example of previously proposed lambda architecture [13] utilized SploutSQL for storing batch views, Apache Cassandra database for speed views, and Apache Storm for merging views that undoubtedly brings an extra implementational effort.

E. Client Application

The client application includes a back-end web application with REST API and a front-end Javascript application with a user dashboard with different types of analytic diagrams and interactive query tools. The back-end application performs queries on unified results in the serving layer. Druid database supports SQL queries through JDBC. The front-end application's dashboard should include time-series charts, histograms, price comparison diagrams, block inspectors, etc. The interactive query tool should offer the easy specification of different filtering, aggregations, and sorting. As back-end supports SQL queries through JDBC, advanced users can write ad-hoc SQL commands directly through the front-end application and then apply different types of analytic diagrams on the result data. On the other hand, novice users can use graphical and simple domain-specific language which allows them to construct a custom query without knowledge of any querying language.

V. DISCUSSION

The main advantage that comes with the use of the proposed architecture is the ability to combine historical data with real-time data and produce results in real-time. Because historical data is extensive (hundreds of gigabytes) and tens of megabytes of new data are produced every second, it can be really hard to maintain the production of results in real-time. Referenced architectures [2], [3] need to collect new data, combine it with historical data, and then rerun algorithms. Those approaches introduce redundant calculations and cannot produce results in real-time. Although the literature is extensive, we could not find related papers that considered lambda architecture as a solution for cryptocurrency data processing. Our proposed architecture solves the problem of real-time results production but brings significant implementational complexity. Batch processing and stream processing need to produce the same result, but work on different kinds of data sources and need to be developed separately, using different technologies. This requires a double effort to implement a big data system using the proposed architecture and also brings new challenges regarding the management and deployment of such a system in different environments. Also, data modeled with Lambda architecture is difficult to migrate or reorganize, therefore it is hard to maintain the system as time progresses and the data model is changing.

Lambda architectural approach brings several challenges regarding design and deployment. It includes issues with implementing two different programming models for batch and stream processing, maintaining code that needs to produce the same result in two complex distributed systems, and programming in distributed frameworks like Kafka, Hadoop, or Spark that could be unexpectedly complex. Because of these challenges that come with Lambda architecture, our further work should be oriented towards an investigation of Kappa architecture and its use in real-time cryptocurrency processing. Kappa architecture [14] was originally presented as an alternative to the lambda approach. It is simpler as it uses the same technology stack to handle both batch processing of historical data and real-time stream processing. Therefore, Kappa architecture is not structurally complex as Lambda architecture and requires fewer resources to develop and maintain. It also needs to be changed only in one place in order to change processing algorithms.

VI. CONCLUSION

The modular design of our architecture enables using heterogeneous big data technologies and frameworks during the implementation of the system components. The proposed architecture provides real-time or near real-time insight into events on the cryptocurrency blockchain networks and cryptocurrency prices in popular exchange markets. These events can be used by crypto market analysts to monitor activities within the cryptocurrency network or make decisions that can be important for their business. Some sorts of insights that can be obtained using cryptocurrency data processing are: mining statistics, tracking large-amount transactions, buying and selling trends, comparison to the other cryptocurrency's transaction frequency, etc. These insights, e.g., comparative statistics between two cryptocurrencies' trends, can help crypto analysts and entrepreneurs to make decisions regarding whether or not transfer funds from one cryptocurrency to another. Another example is tracking large-amount transactions that can be useful for detecting suspicious or illegal activities, e.g., terrorism financing.

Our proposed lambda architecture brings complexity and robustness, but it involves suitable individual components for cryptocurrency data processing and analytics. The master dataset uses HDFS which is distributed, scalable, portable, and fault-tolerant. Data is safely stored in HDFS and because it is distributed, reading data from it can be done in parallel.

Parallel reading enables better performance in batch processing because hundreds of megabytes are loaded and processed each second. Batch processing procedures are implemented through Spark SQL jobs that are performed on data stored in HDFS. Spark SQL is an abstraction above the Spark computational engine which brings parallel batch processing and ease of use because queries are written in conventional SQL-like language. The speed layer can utilize Kafka Streams API or Spark Streaming. Both stream processing libraries enable scalable, high throughput processing capability on streaming data. After being processed through a batch or speed layer, resulting data is saved in the Serving layer implemented using Apache Druid. Due to Druid's capability of both real-time and batch ingestion, there is no need for implementation of an extra procedure for merging real-time and batch results.

Utilizing the lambda architectural approach and choosing its components so that they offer a balance between performance and implementational complexity provides an excellent real-world example for the big data Architectures course. It can help students to better understand real-time big data architectures and become familiar with leading open-source technologies and frameworks that can be efficiently used in such architectures to implement modular and extensible systems.

ACKNOWLEDGMENT

This paper has been supported by the Ministry of Education, Science and Technological Development through the project no. 451-03-68/2020-14/200156: "Innovative scientific and artistic research from the FTS (activity) domain."

REFERENCES

[1] Nakamoto, Satoshi, and A. Bitcoin. "A peer-to-peer electronic cash system." Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf (2008).

[2] Laskowski, Marek, and Henry M. Kim. "Rapid prototyping of a text mining application for cryptocurrency market intelligence." 2016 IEEE 17th International Conference on Information Reuse and Integration (IRI). IEEE, 2016.

[3] Dulău, Tudor-Mircea, and Mircea Dulău. "Cryptocurrency–Sentiment Analysis in Social Media." Acta Marisiensis. Seria Technologica 16.2 (2019): 1-6.

[4] Renani, Fatemeh, Jaskaran Kaur Cheema, and Mohammad Mazraeh. "Real-Time Cryptocurrency Analysis."

[5] Jiang, Zhengyao, and Jinjun Liang. "Cryptocurrency portfolio management with deep reinforcement learning." 2017 Intelligent Systems Conference (IntelliSys). IEEE, 2017.

[6] De Mauro, Andrea, Marco Greco, and Michele Grimaldi. "What is big data? A consensual definition and a review of key research topics." AIP conference proceedings. Vol. 1644. No. 1. American Institute of Physics, 2015.

[7] Katal, Avita, Mohammad Wazid, and Rayan H. Goudar. "Big data: issues, challenges, tools and good practices." 2013 Sixth international conference on contemporary computing (IC3). IEEE, 2013.

[8] Demchenko, Yuri, Cees De Laat, and Peter Membrey. "Defining architecture components of the Big Data Ecosystem." 2014 International Conference on Collaboration Technologies and Systems (CTS). IEEE, 2014.

[9] Kaisler, Stephen, et al. "Big data: Issues and challenges moving forward." 2013 46th Hawaii International Conference on System Sciences. IEEE, 2013.

[10] Hassani, Hossein, Xu Huang, and Emmanuel Silva. "Big-Crypto: Big data, blockchain and cryptocurrency." Big Data and Cognitive Computing 2.4 (2018): 34.

[11] Zheng, Zibin, et al. "An overview of blockchain technology: Architecture, consensus, and future trends." 2017 IEEE international congress on big data (BigData congress). IEEE, 2017.

[12] Marz, Nathan, and James Warren. Big Data: Principles and best practices of scalable realtime data systems. Manning Publications Co., 2015.

[13] Hasani, Zirije, Margita Kon-Popovska, and Goran Velinov. "Lambda architecture for real-time big data analytic." ICT Innovations (2014): 133-143.

[14] Kreps, Jay. "Questioning the Lambda Architecture." O'Reilly Media, 2 July 2014. www.oreilly.com, https://www.oreilly.com/radar/questioning-the-lambda-architecture/.