Hephaistos: A Management System for Massive Order Book Data from Multiple Centralized Crypto Exchanges with an Internal Unified Order Book

Robert Henker[1], Daniel Atzberger[2], Jan Ole Vollmer, Willy Scheibel[2], Jürgen Döllner[2], Markus Bick[3]
[1]*XU Exponential University, Potsdam, Germany*
[2]*Hasso Plattner Institute, Digital Engineering Faculty, University of Potsdam, Germany*
[3]*ESCP Business School, Berlin, Germany*
mail@roberthenker.com, janolevollmer@web.de, mbick@escp.eu
{daniel.atzberger, willy.scheibel, juergen.doellner}@hpi.uni-potsdam.de

***Abstract*—Offers to buy and sell cryptocurrencies on exchanges are collected in an order book as pairs of amount and price provided with a timestamp. Contrary to tick data, which only reflects the last transaction price on an exchange, the order book reflects the market's actual price information and the available volume. Until now, no system has been presented that can capture many different order books across several markets. This paper presents Hephaistos, a system for processing, harmonizing, and storing massive spot order book data from 22 centralized crypto exchanges and 55 currency pairs. After collecting the data, Hephaistos aggregates several order books in a so-called Unified Order Book, which is the foundation for a Smart Order Routing algorithm. As a result an order is splitted across several exchanges, which results in a better execution price. As component of a high-frequency trading system, Hephaistos captures 32 % of the total daily spot trading volume. We provide examples with data from two exchanges that show that the Smart Order Routing algorithm significantly reduces the slippage.**

***Index Terms*—Cryptocurrencies, Centralized Exchanges, Order Book Data, Big Data, Stream Processing**

I. Introduction

Since the launch of the first and most famous cryptocurrency, Bitcoin, more than 9 000 other cryptocurrencies have been created that can be traded on more than 240 specialized crypto exchanges [1]. In contrast to the frequently used closing prices, so-called tick data, which reflect the price at which the last transaction was executed in the past, order book data also make available the prices at which buyers and sellers would now buy and sell, but whose orders have not yet been executed due to a lack of matching complementary orders and will not necessarily have to be executed in the future [2]. The order book totals the volumes of all existing buy and sell orders at a given price and compares the totals sorted by amount as supply and demand. Besides fiat currencies such as EUR and USD, other cryptocurrencies can be used for trading a cryptocurrency, thus resulting in a variety of pairings. The exchange maps the currently available price from the offered, so-called maker orders, lowest selling price and the highest buying price. If a buyer, so-called taker orders, buys equal to or higher than the lowest offered selling price or places an order equal to or lower than the highest offered buying price, the crypto exchange forms a transaction from the highest possible matching volume of the maker and taker orders. This transaction's price is then added as the latest tick to the tick data. In illiquid market phases or in the case of larger transactions, close observation of the order book is necessary, as the actual price available in the order book may deviate significantly from the last tick price [3].

Order Book Data is an example of Big Data using a taxonomy following Jin et al. [4]:

Volume: The collected order books within three years are stored as a multivariate dataset with a total size of 55TB.

Velocity: At liquid exchanges, an order book has an update frequency of only a couple of milliseconds.

Veracity: Order book entries sometimes contain non-sensible entries that have to be detected and filtered.

Variety: Some exchanges provide additional data besides the amount and price in an order book, e.g., notional volume, time of entry, time of last change, or order type.

Value: Knowledge of historical order book data is necessary for evaluating quantitative investment strategies in back-tests [5], analyzing market structures [3], [6], or locating fraud activities [7], [8].

These characteristics of order book data cause, among other factors, a high level of technical complexity in their processing, storing, and analysis. To the best of our knowledge, there is no free service that provides real-time access to order books across several exchanges in a unified data scheme. Therefore our goal was to design and implement a customized solution.

In this paper, we present the design and technical implementation of a system, named *Hephaistos*, that collects data from the order books of multiple exchanges by accessing their respective public APIs. After checking each entry in the order book for errors, the accepted entries are stored in a uniform multivariate data scheme. In a further step the single order books are aggregated in a single Unified Order Book

979-8-3503-1019-1/23/$31.00 ©2023 IEEE

(UOB). By applying a Smart Order Routing (SOR) algorithm, an incoming order is split across several exchanges to achieve a better execution price. To summarize, our work makes the following contributions:

1) We built a fully working system that is integrated in a high-frequency trading system, which captures 32 % of the total daily spot trading volume.
2) We present design considerations and details on its implementation.
3) We present a UOB that aggregates the collected data within a single data structure, which allows the application of a SOR algorithm to reduce slippage.

The remainder of this work is structured as follows: In Section II we present existing works for mining financial data for the cryptocurrency domain. The concept of our system, our data base scheme, and implementation details are described in Section III. As an use case for storing order books across several exchanges, we present a SOR algorithm together with an example for reducing slippage in Section IV. Our results are discussed in Section V. We conclude this work and point out directions for future work in Section VI.

II. Related Work

Our consideration of the related work comprises three parts. First, we present papers that ignore order books and include only tick data in their considerations. However, different studies show that liquidity is a crucial aspect when investing in any asset, particulary in digital assets or cryptocurrencies, thus requiring knowledge about order book data [3]. We then proceed with a presentation of existing solutions for crawling order book data from crypto exchanges, and finally present different applications that rely on order book data.

A. Applications relying on tick data

Most related work focusses on tick data when analyzing cryptocurrencies. Ho et al. studied trading strategies for 23 cryptocurrencies, that are derived from candlestick patterns [9]. Their analysis is limited to aggregated daily price data and only requires the past opening, closing, high, and low prices, but ignores intraday price fluctuation. McNally et al. trained recurrent neural networks for price prediction based on historical daily closing prices and compared them with the classical *ARIMA* model for time series analysis [10]. Another popular research direction is the combination of tick data with text data obtained from *Twitter* for predicting price movements [11], [12].

None of the presented methods considers the underlying order book in their studies. Accordingly, only explicit costs are taken into account in backtests, e.g., trading fees of the exchanges or clearing fees, whereas implicit costs are neglected. However, considering the order book is indispensable, especially in the operational implementation of a trading strategy, since dealing with the slippage effect during the execution of an order is a challenge. The slippage effect occurs when one sell/buy order is executed against multiple buy/sell orders in the order book. As a result, the effective price where the resulting trades were executed deviate significantly from the initially assumed price. A study on the impact of implicit costs measured by several liquidity measures for different crypto exchanges is provided by Angerer et al. [3]. To adress this issue, we propose a SOR algorithm that reduces the implicit costs, which is a common technique in traditional financial markets [13].

B. Existing systems

Existing work on analysis and visualization of crypto data is limited to a small amount of data at a time, usually comprising only one currency pair and collected within a short time period such as one year. Moreover, the papers do not present any software implementation capable of managing order book data as Big Data. An exception is made by Dilbagi, who presents a hybrid persisted cloud approach for collecting order book data from *Binance* [14].

Another system architecture for collecting order book data from crypto exchanges using a GPU-accelerated processing pipeline, was proposed by Burján and Gyires-Tóth [15]. Their system collects data from *Coinbase* and stores the data for further analysis using neural networks. Our work goes further and collects order book data from more than one exchange and several currency pairs. This extension requires a more advanced system for processing, harmonizing, and storing the data. In our work, we therefore focus on the development of an infrastructure that is capable of creating a comprehensive data basis of order books for many cryptocurrency pairs from multiple exchanges over a long period of time that enables downstream analyses. To the best of our knowledge, we are the first to present a work in that direction.

C. Applications reyling on order book data

Puljiz et al. presented a work for detecting fraud activities in the cryptocurrency domain [6]. Their work relies on the order book of the BTC/USD pair collected at one exchange within 300 hours. In their work, the authors apply a continuous-time stochastic model to investigate the dynamics underlying an order book. A main result of their work was the evidence of widespread use of frontrunning, which is one among various market manipulation techniques [7], [16]. Front running refers to exploiting insider information in securities transactions by a stockbroker. It is assumed, for example, that the insider first buys these securities for his own portfolio before placing a large buy order to profit from the price increase of the subsequent order.

Besides fraud detection, the analysis of quantitative trading strategies is a relevant application relying on order book data. Raheman et al. presented a market-making strategy that uses multiple independent agents, each with different strategies [5]. Their strategy increased returns in backtests on two order books and in a real-world setting.

III. Architecture

Figure 1 outlines the components of our system and their distribution across servers. The order book reconstruction and

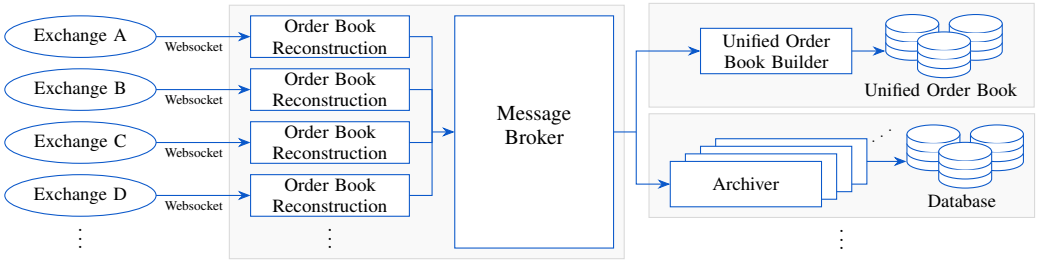

Fig. 1. System overview with the different components, instances, and their distribution across servers (grey rectangles).

archiving components are implemented in Python and we use two independent PostgreSQL databases on separate servers for redundancy.

A. Order Book Reconstruction

Most exchanges offer a Websocket or FIX API to receive updates to the order book incrementally. The typical process starts with receiving or fetching a snapshot of the complete order book and then subscribing to a stream of incremental updates, where each update contains only the price levels that have changed since the preceding update. Three update cases can occur: (1) add a new price level, (2) change the volume of an existing price level, (3) delete a price level, though often the same representation of a $(price, volume)$-tuple is used for all three cases with a volume of 0 denoting the deletion of a level.

It is up to the client to correctly reconstruct the order book from the stream of incremental updates by applying each update to their local state and ensure their local order book is fully in-sync with the source exchange's order book. There are different options to verify the integrity of the reconstructed order book, whose availability differ by exchange:

Update timestamps are the most basic variant and allow verifying that updates have at least been received in the correct order.

Update sequence numbers can be considered extension of update timestamps and are expected to be strictly incremental, thus additionally allow the detection of missing updates.

Checksums are the most comprehensive approach that not only allows verifying the correct receiving of updates, but also the correctness of the reconstructed order book. They are computed over the first n levels of the order book after applying the respective update.

Our system applies all three integrity checks as far as provided by the different exchanges. The reconstruction component of our system encapsulates all exchange-specific message parsing and API behavior, such as the handling of maximum order book depth, where some exchanges send explicit deletions for levels that exceed the maximum depth due to intermediately inserted levels, while others rely on the client to discover this. The reconstruction component outputs streams of fully reconstructed order book snapshots in a harmonized format.

Our system runs one or more reconstruction processes per connected exchange to distribute the computational workload over multiple CPU cores, all of which feed the resulting snapshots into a central message broker.

B. Message Broker

We introduced a central *Apache Kafka* message broker following the publisher-subscriber model to allow flexible scaling-out and distribution across servers on both ends. This avoids n:m connections between publishers (order book reconstruction) and subscribers (such as archiving) as well as having to perform the reconstruction of the same order book multiple times in different processes. Kafka offers low-enough end-to-end latency as well as an intermediate storage with guaranteed message ordering, which can act as cache and bridge load spikes as well as outages of the database and archiving services, thus ensuring completeness of the archived data. We use separate Kafka topics for each trading pair on each exchanges to allow consumers to selectively subscribe to required markets only, thus reducing the overall message load on the broker as well as the consumer.

C. Archiving

We archive all received order book data without any aggregation or reduction for later analysis. The archiving component subscribes to all Kafka topics and encodes the received data into the database storage format described in section III-D, which involves deriving incremental changes again from the stream of snapshots in order to reduce the required storage space. The archiver keeps the current state in memory and only outputs a new database entry once a level has been removed or replaced and thus the $valid\_until$ timestamp is know. This reduces the number of write operations to the database and ensures consistency compared to saving every level as soon as it arrives and filling in the $valid\_until$ timestamp later. The disadvantage of this approach is that it can take considerable time until the full snapshot for a particular point in time is available in the database as levels further down in the order book change less frequently.

The resulting table rows are committed in batches to reduce the transaction processing overhead in the database. Together with each batch, the archiving service stores the latest message id whose data is included in the batch for every Kafka topic in the same transaction. This ensures that in the event of a crash or restart, the archiving service can find the exact place in the order book stream where the archiving needs to continue without any data loss or integrity violations.

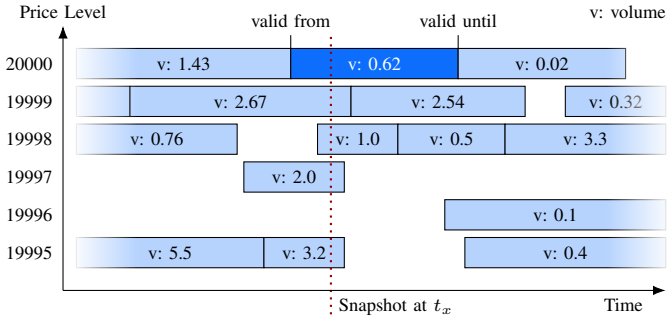

Fig. 2. Conceptual visualization of order book level changes over time.

The workload of the archiving can be easily distributed over multiple processes along different Kafka topics since encoding and database writes for different streams do not conflict.

D. Database Storage

Our database storage concept combines three aspects to maximize space efficiency while maintaining acceptable speed for typical workloads.

Validity Timestamps: Figure 2 shows a contrived example of an order book and the changes to its levels over time. Each entry for a specific price level has a specific volume and exists for a specific timeframe until it is either replaced with an entry of a different volume or deleted. We denote the lifetime of such an entry using two timestamps: $valid\_from$ for when an entry first enters the order book and $valid\_until$ for when the entry is replaced or removed. The complete snapshot of the order book at any point in time t_x consists of all entries with $valid\_from \leq t_x$ and $valid\_until > t_x$.

The key idea to our storage format is to store each entry as a separate row including the $valid\_until$ timestamp. This allows directly fetching the order book snapshot for any point in time without requiring a starting snapshot and replaying incremental updates as received by the exchanges while maintaining the space efficiency compared to storing full order book snapshots after every update.

Table Partitioning: Based on observations of use cases and queries run on the data, we determined that the majority of queries involve data from only small subset of markets (most often just one) and never merge bids and asks. This allows the use of *table partitioning* to reduce the query response time [17]. When using magnetic hard disks, the disk response time is a major factor in the total time required to filter a large table. To reduce the amount of data that needs to be searched and therefore loaded from disk, we use completely separate tables for bids and asks. Each of these two tables further uses Postgres' built-in table partitioning mechanism to physically store data from different markets at separate locations on the disk. Queries can directly access the required partition or use the parent table with a filter for the market, in which case the query planner automatically determines the appropriate partition(s). In both cases, the amount of data that needs to be loaded from disk is drastically reduced, thus reducing the query response time accordingly. In addition, this speeds up sequential accesses of specific tables (such as occurs when exporting data for a timerange or replaying it during backtesting) even further due to the improved spatial locality on disk.

BRIN Indexes: The search for a particular point in time or timerange using the $valid\_from$ and $valid\_until$ fields is a common filter criterion in our queries. While it is possible to use traditional b-tree indexes to accelerate the search of a table for a particular value, they generally require as much disk space as the target column(s) since they need to contain the value of every row and thus exhibit a considerable space overhead for tables with a few columns but many rows.

Due to the way the archiver works, the tables are effectively *insert-only* tables, i.e., data is only ever inserted, but never modified or deleted. Further, on a large scale view, the order of rows inserted into the tables and thus their order on disk correlates with the $valid\_from$ and $valid\_until$ timestamps. The *Block Range Index (BRIN)* index is a highly specialized index for exactly such cases where the target column correlates with the order on disk [18]. It divides the set of table rows into blocks and stores the value range (minimum and maximum) for the target columns in every block. The lookup consist of a linear search of block summaries for blocks whose value range covers the value in question, followed by linear searches of every block found for the exact matches. This type of index provides sufficient speed-up for our use case with minimal space overhead for storing the index data.

E. Unified Order Book Builder

The key idea of a UOB is that it includes price levels and the related volumes from several exchanges that list a given pair. Each price level of the UOB includes the exchange it originated from in addition to price and volume. If multiple exchanges offer the same price level, they are not aggregated but included separately to allow subsequent separation again. The UOB thus diverges from traditional representations in that it is only monotonically in/decreasing, but not strictly monotonically, i.e., if two different exchanges contain two levels with the same price they will occur as separate entries within the UOB. An example of the UOB originating from two order books is shown in Table I.

The building of the UOB is a separate component. It is implemented as a stream processor, i.e., a separate process receives individual order book updates from Kafka, builds the UOB, and feeds the result back into Kafka. This has the advantage that the calculation is done only once, and multiple engine instances may use the same result. However, it has the drawback that it results in higher latency and overall computation effort if only one instance uses the result

IV. Smart Order Routing

In illiquid market phases or when placing large orders, several levels of the order book are required for execution. The associated implicit costs lead to a higher average purchase or lower average selling price. In the following, we present a SOR algorithm that splits an order and distributes it across multiple exchanges to minimize the implicit costs. The SOR

TABLE I
FIRST TEN LEVELS OF THE ASK SIDE OF TWO ORDER BOOK SNAP SHOTS FROM *CEXIO* AND *Kraken* TOGETHER WITH THE RESULTING UOB.

CEXIO		Kraken		Unified Order Book		
Price	**Volume**	**Price**	**Volume**	**Price**	**Volume**	**Exchange**
20 078.3 $	0.001 BTC	20 092.6 $	0.004 BTC	20 078.3 $	0.001 BTC	CEXIO
20 132.3 $	0.410 BTC	20 098.9 $	0.647 BTC	20 092.6 $	0.004 BTC	Kraken
20 132.4 $	0.573 BTC	20 105.2 $	0.619 BTC	20 098.9 $	0.647 BTC	Kraken
20 132.6 $	0.900 BTC	20 115.3 $	0.486 BTC	20 105.2 $	0.619 BTC	Kraken
20 137.7 $	0.498 BTC	20 136.2 $	0.400 BTC	20 115.3 $	0.486 BTC	Kraken
20 187.9 $	1.000 BTC	20 191.0 $	5.289 BTC	20 132.3 $	0.410 BTC	CEXIO
20 321.9 $	0.148 BTC	20 200.9 $	0.124 BTC	20 132.4 $	0.573 BTC	CEXIO
20 322.0 $	2.000 BTC	20 228.8 $	0.100 BTC	20 132.6 $	0.900 BTC	CEXIO
20 500.0 $	0.250 BTC	20 238.1 $	0.003 BTC	20 136.2 $	0.400 BTC	Kraken
20 966.4 $	0.001 BTC	20 254.8 $	0.001 BTC	20 137.7 $	0.498 BTC	CEXIO

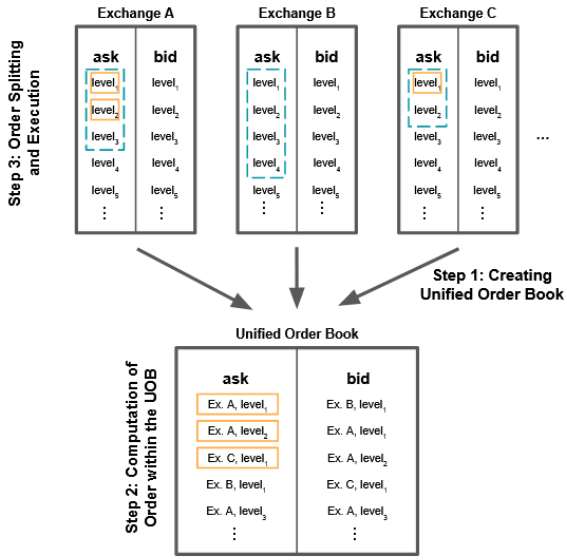

Fig. 3. Procedure of the SOR Algorithm. An order might require several levels in an order book to be executed. The blue boxes group the required levels within a single order book that belong to such a trade. The orange boxes indicate the levels of the splitted order.

algorithm follows three steps as shown in Figure 3. First, the UOB is created for the exchanges under consideration. Secondly, the splitted orders are computed by considering the UOB. In the third step, the single orders are executed on the single exchanges.

To illustrate the effectiveness of our approach, we consider the concrete example from Table I. In Table II the average buy prices and the slippage, i.e., the relative distance to the first order book level, are summarized. By construction, the SOR algorithm consistently achieves the lowest average price. The difference to the respective price on one of the two alternative exchanges increases with growing investment volume and amounts to up to 0.38 %. The UOB also has the smallest best ask price, which means that for small investment volumes, the slippage on a single exchange, specifically Kraken in this case, is smaller. However, from an investment volume of 3.0 BTC, the relative slippage in the UOB is the smallest. The difference increases with growing investment volumes, For an investment of 5.0 BTC, the implied cost is 158 % higher on CEXIO and 28 % higher on Kraken. It is worth mentioning that we are only looking at two exchanges here. With a growing number, the reduction of the slippage effect by the SOR can be expected to increase significantly.

V. RESULTS & DISCUSSION

The Hephaistos system is tailored to handle order book data from several crypto exchanges and is effective regarding this approach with the current setup. Currently, it collects the order books of 22 exchanges and 55 currency pairs, representing 1 210 individual markets (unique pairs per exchange). According to *CoinMarketCap* those markets represent 32 % of the total daily spot trading volume.In September 2022 alone, the 1 210 unique markets generated 32 043 446 130 price updates, which corresponds to 1.7 TB additional storage volume in the system.

However, the trading of traditional assets, e.g., stocks, and alternative assets, e.g., commodities or energy, is also organized via order books. Providing this data is an essential part of the business model of traditional exchange operators. An extension of Hephaistos to more liquid markets will have impact regarding the scalability of the architecture and the adaptability to other exchanges and their APIs.

Scalability: Regarding scalability, the processing of a larger trade throughput is required. Using parallelization across multiple CPUs, the system can scale up on order book reconstruction accordingly. Archiving only requires the addition of more storage.

Extensibility to other Exchanges: The respective exchanges allow access to the order books of the respective markets via an API interface. These APIs are handled using corresponding new implementations of order book reconstructions and therefore do not require any architectural changes. Although each exchange can communicate more data in their order book updates, we don't expect data structure mismatches with our unified storage scheme.

VI. CONCLUSIONS

Centralized exchanges enable the trading of cryptocurrencies. Buy and sell orders are managed in the order book and thus

TABLE II
AVERAGE PRICE FOR BUYING ONE BITCOIN FOR DIFFERENT ORDER VOLUMES. THE SLIPPAGE MEASURES THE RELATIVE DISTANCE BETWEEN THE RESULTING AVERAGE PRICE AND THE FIRST LEVEL IN THE RESPECTIVE ORDER BOOK.

	CEXIO		Kraken		Unified Order Book	
Volume	**Avg. Price**	**Slippage**	**Avg. Price**	**Slippage**	**Avg. Price**	**Slippage**
0.5 BTC	20 132.21 $	0.27 %	20 098.85 $	0.03 %	20 098.80 $	0.10 %
1.0 BTC	20 132.31 $	0.27 %	20 101.07 $	0.04 %	20 101.05 $	0.11 %
1.5 BTC	20 132.41 $	0.27 %	20 104.00 $	0.05 %	20 103.97 $	0.13 %
2.0 BTC	20 132.75 $	0.27 %	20 109.37 $	0.08 %	20 108.87 $	0.15 %
2.5 BTC	20 136.10 $	0.29 %	20 122.28 $	0.15 %	20 113.57 $	0.18 %
3.0 BTC	20 144.74 $	0.33 %	20 133.73 $	0.20 %	20 116.73 $	0.19 %
3.5 BTC	20 155.42 $	0.38 %	20 141.91 $	0.25 %	20 119.00 $	0.20 %
4.0 BTC	20 176.25 $	0.48 %	20 148.05 $	0.28 %	20 121.02 $	0.21 %
4.5 BTC	20 192.44 $	0.57 %	20 152.82 $	0.30 %	20 122.86 $	0.22 %
5.0 BTC	20 205.40 $	0.63 %	20 156.64 $	0.32 %	20 128.98 $	0.25 %

form the basis for price formation. The retrieval of order book data across multiple exchanges and currency pairs, poses great challenges for its technical implementation. To overcome these challenges, we developed Hephaistos, a system and infrastructure for processing, harmonizing, and storing massive order book data. The system is able to reconstruct an order book in real time and allows to archive this data in an UOB. As one possible application of the UOB, we presented a SOR algorithm to execute one order across multiple marketplaces resulting in multiple transactions to counteract the slippage effect and thus achieving a better execution price.

Our next step is to investigate to what extent the use of the SOR helps to improve the return on quantitative investment strategies. For this purpose, simple trading strategies, e.g., based on chart signals, will be simulated. In this context, the necessity of knowledge of the order book can also be demonstrated.

One of the observations on the UOB is that a negative spread might occur, i.e., the best ask price can be lower than the best bid price on another exchange. Such a situation offers an arbitrage opportunity in which an asset can be bought on one exchange and simultaneously sold on another exchange with profit. It is necessary to investigate to what extent the market data show such theoretical opportunities and if these can be actually exploited.

Another possible direction for future work is to apply existing approaches relying on order book data, e.g., fraud detection, on a larger dataset. Also, a comparison to the state of UOB and SOR in traditional financial markets, e.g. in terms of technical architecture, data throughput, and latency, would be beneficial.

REFERENCES

[1] CoinMarketCap. Top cryptocurrency spot exchanges. 2023. URL: coinmarketcap.com/rankings/exchanges/.

[2] Martin D Gould, Mason A Porter, Stacy Williams, Mark McDonald, Daniel J Fenn, and Sam D Howison. Limit order books. *Quantitative Finance*, 13(11):1709–1742, 2013.

[3] Martin Angerer, Marius Gramlich, and Michael Hanke. Order book liquidity on crypto exchanges. In *Proceedings of the 3rd Crypto Asset Lab Conference*, CAL '21. Crypto Asset Lab, 2021.

[4] Xiaolong Jin, Benjamin W Wah, Xueqi Cheng, and Yuanzhuo Wang. Significance and challenges of big data research. *Big Data Research*, 2(2):59–64, 2015.

[5] Ali Raheman, Anton Kolonin, Ben Goertzel, Gergely Hegyközi, and Ikram Ansari. Architecture of automated crypto-finance agent. In *Proceedings of the International Symposium on Knowledge, Ontology, and Theory*, KNOTH '21, pages 10–14. IEEE, 2021.

[6] Mate Puljiz, Stjepan Begušic, and Zvonko Kostanjcar. Market microstructure and order book dynamics in cryptocurrency exchanges. pre-print, URL: www.bib.irb.hr/952865, 2018.

[7] Felix Eigelshoven, Andre Ullrich, and Douglas A Parry. Cryptocurrency market manipulation: A systematic literature review. In *Proceedings of the International Conference on Information Systems*, ICIS '21. AIS, 2021.

[8] Friedhelm Victor and Andrea Marie Weintraud. Detecting and quantifying wash trading on decentralized cryptocurrency exchanges. In *Proceedings of the Web Conference 2021*, WWW '21, pages 23–32. ACM, 2021.

[9] Kin-Hon Ho, Tse-Tin Chan, Haoyuan Pan, and Chin Li. Do candlestick patterns work in cryptocurrency trading? In *Proceedings of the International Conference on Big Data*, BigData '21, pages 4566–4569. IEEE, 2021.

[10] Sean McNally, Jason Roche, and Simon Caton. Predicting the price of bitcoin using machine learning. In *Proceedings of the 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, PDP '18, pages 339–343. IEEE, 2018.

[11] Jethin Abraham, Daniel Higdon, John Nelson, and Juan Ibarra. Cryptocurrency price prediction using tweet volumes and sentiment analysis. *SMU Data Science Review*, 1(3), 2018.

[12] Shubhankar Mohapatra, Nauman Ahmed, and Paulo Alencar. KryptoOracle: A real-time cryptocurrency price prediction platform using twitter sentiments. In *Proceedings of the International Conference on Big Data*, BigData '19, pages 5544–5551. IEEE, 2019.

[13] Xiaoyun Wang and Tu Lai Huan. Bnp paribas: Equity smart order router. *Electionic Projects Collection*, 2010.

[14] Arsh Dilbagi. Infrastructure and techniques to collect data and detect market manipulation on crypto-exchanges. Master's thesis, Operations Research and Financial Engineering, Princeton University, New Jersey, USA, 2021.

[15] Viktor Burján and Bálint Gyires-Tóth. Gpu accelerated data preparation for limit order book modeling. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 385–397. Springer, 2020.

[16] Massimo Bartoletti, Barbara Pes, and Sergio Serusi. Data mining for detecting bitcoin ponzi schemes. In *Proceedings of the Crypto Valley Conference on Blockchain Technology*, CVCBT '18, pages 75–84. IEEE, 2018.

[17] PostgreSQL. Documentation table partitioning. 2023. URL: postgresql.org/docs/12/ddl-partitioning.html.

[18] PostgreSQL. Documentation brin indexes. 2023. URL: www.postgresql.org/docs/12/brin.html.