Uppsala Student Project 2017

Financial Surveillance Using Big Data

Project Specification

Industry representatives

Fredrik Lydén Gustaf Gräns Gustav Tano

Scila AB

2 Summary

The scope of this project is to create an application that reads the extreme amounts of financial data that the Scila system produces and provide cloud based tools to:

- Process the data in a cloud environment
- Batch/ad-hoc visualizations/reports
- Batch-oriented market abuse pattern detection
- Anomaly detection using Machine Learning

3 Introduction

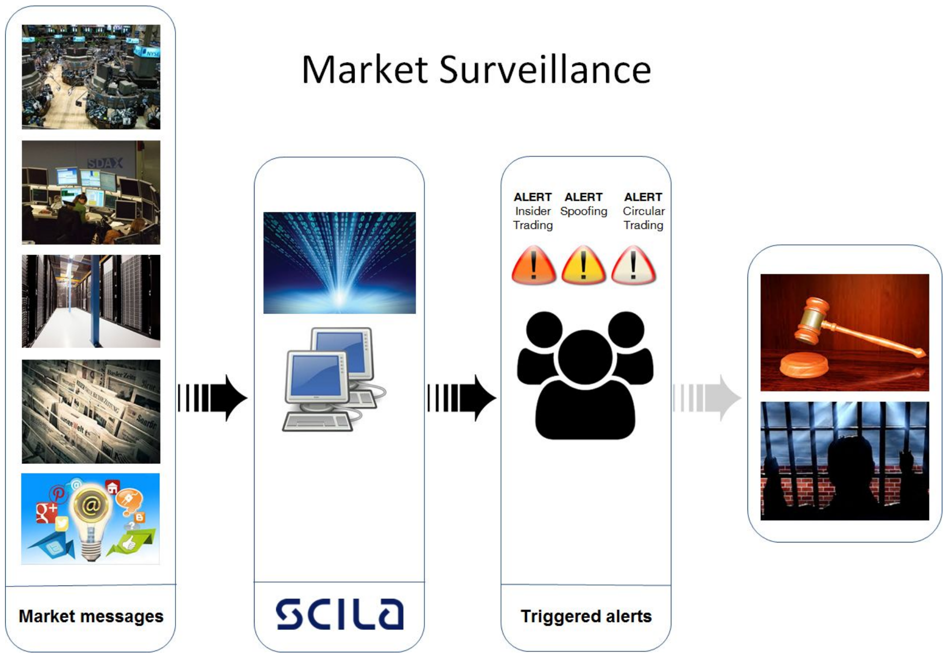
Scila Surveillance is a market surveillance solution for exchanges, trading participants and regulators that applies modern technology to obtain an early detection of market abuse and data to create presentable evidence. The solution covers all asset classes and market models and has been deployed for over 30 clients in 13 countries since 2008.

All data that the Scila application consumes are written to disc in a modified JSON format. The data consists of events and reference data for instruments, exchanges and market participants. The Scila system creates vast amounts of data. At some customers the system can consume more than 30 billion messages per day and million events per second. The main functionality of the Scila application is to stream data real time and surveil for possible market abuse. The stored data is used to investigate potential breaches using graphical views for analyzation and replay, event by event, in the Scila Client. The data can also be used for extensive reporting.

The goal of this project is to provide a proof of concept how Apache Spark can be used to process and present data that the Scila system produces. This is done by creating an application that can process large volumes of Scila data and to provide tools to visualize and run pattern detection on it.

4 Background

The Scila system analyzes a wide range of Market Messages that originate from Financial Market Places and surrounding systems such as Orders, Trades, Reference Data, Reference Prices, News, Positions and State Changes. The messages are analyzed in real-time. The system triggers Alerts when suspicious behaviour is detected. This project will focus on two message types; orders and trades. The data produced by a Scila System can be analyzed in a cloud-based environment.



4.1 Orders

An order is an instruction to buy or sell on a trading venue such as a stock market, bond market, commodity market, or financial derivative market. These instructions can be simple or complicated. Two of the standard Order Types are [Ref 2]:

- Market Order - A market order is a buy or sell order to be executed immediately at current market prices. As long as there are willing sellers and buyers, market orders are filled. Market orders are therefore used when certainty of execution is a priority over price of execution. A market order is the simplest of the order types. This order type does not allow any control over the price received. The order is filled at the best price available at the relevant time. In fast-moving markets, the price paid or received may be quite different from the last price quoted before the order was entered.
- Limit Order - A limit order is an order to buy a security at no more than a specific price, or to sell a security at no less than a specific price (called "or better" for either direction). This gives the trader (customer) control over the price at which the trade is executed; however, the order may never be executed ("filled"). Limit orders are used when the trader wishes to control price rather than certainty of execution. A buy limit order can only be executed at the limit price or lower. For example, if an investor wants to buy a stock, but doesn't want to pay more than $20 for it, the investor can place a limit order to buy the stock at $20. By entering a limit order rather than a market order, the investor will not buy the stock at a higher price, but, may get fewer shares than he wants or not get the stock at all. A sell limit order is analogous; it can only be executed at the limit price or higher. A limit order that can be satisfied by orders in the limit book when it is received is marketable. For example, if a stock is asked $86.41 (large size), a buy order with a limit of $90 can be filled right away. Similarly, if a stock is bid $86.40, a sell order with a limit of $80 will be filled right away. A limit order may be

partially filled from the book and the rest added to the book. Both buy and sell orders can be additionally constrained. Two of the most common additional constraints are fill or kill (FOK) and all or none (AON). FOK orders are either filled completely on the first attempt or canceled outright, while AON orders stipulate that the order must be filled with the entire number of shares specified, or not filled at all. If it is not filled, it is still held on the order book for later execution.

4.2 Order Book

An order book is a list of buy and sell orders for a specific security or financial instrument, typically organized by price level and sorted on time. The order book lists the number of shares or lots being bid or offered at each price point, or market depth [Ref 1].

In today's financial markets, order books are primarily electronic and updated automatically by computers or algorithms. However, some over-the-counter (OTC) markets are still traded via phone or in specialized auctions. In those cases the order book can be maintained and updated manually by traders or dealers.

An order book can also be defined as a "tradeable entity", since the same instrument (e.g. stock) can be traded in different order books. For example, Ericsson stock can be traded on different markets operated by different exchanges, i.e. in different order books. Consequently, the term order book refers to the tradable entity and not the instrument or security itself.

Depending on the market and asset class, different types of order books exist:

Transparency

Lit order book - transparent, which means that participants can see orders in the order book, i.e. price and volume

Dark order book - non-transparent, which means that participants are not able to see other orders than their own

Lit/dark hybrid - semi-transparent, typically participants can see aggregated price and volume information, but not specific market depth or volume

Matching logic

Auto-matching - orders are matched continuously into trades by the trading engine ("matching engine"). Most common matching logic is by price and time.
Auto-matching is most common for liquid markets such as equities, options and futures.

Call Auctions - orders are collected and matched at one given moment. The matching process is called to uncross the book. Auctions can be used to match orders for less liquid markets. Reasons to use an call auction includes:

1. Establish an equilibrium price before moving to continuous/automatch trading
2. Concentrate liquidity at certain time intervals, for example opening or closing of the market
3. Establish a benchmark price

4.3 Trade

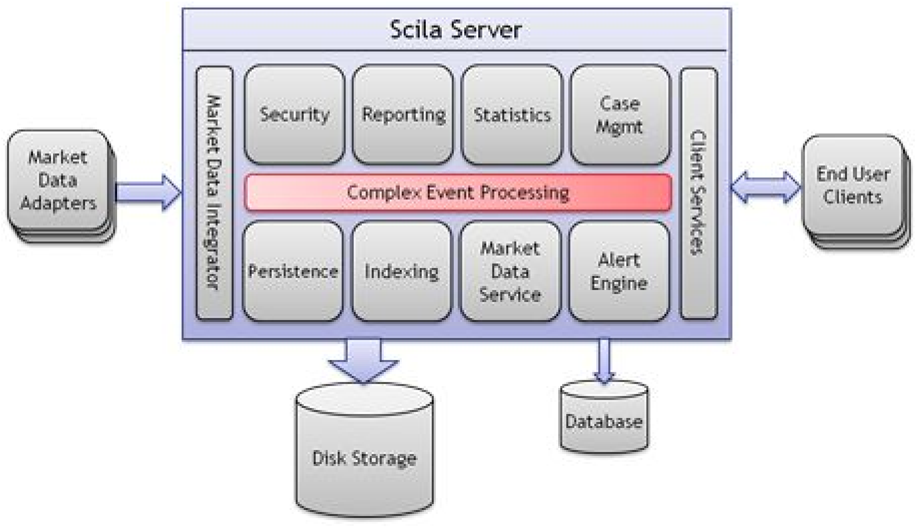
In financial markets, trading refers to the buying and selling of securities, such as the purchase of stock on the floor of the New York Stock Exchange. A trade is two orders which has been matched and executed. Information usually provided to the Scila system in a trade message are: participant information, originating orders, price volume, time. [Ref 3].

5 Architectural components

5.1 Scila high-level technical architecture

The Scila Surveillance architecture is based around the ability to capture and analyze large amounts of events. From a high-level perspective the Scila Surveillance system is composed of one or multiple servers to which multiple clients connect. The server(s) is responsible for collecting, processing and storing the incoming market data information and to distribute the data to the clients connected to the server. The client application is used by the surveillance staff to monitor the marketplace, either in real-time or in batch-mode.

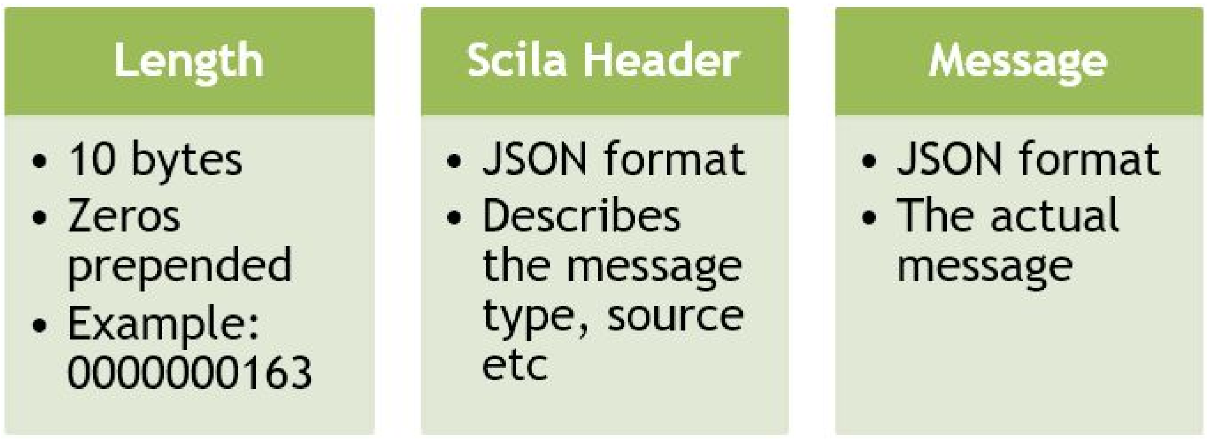
High-level architecture of Scila Surveillance.



5.2 Message Format

The message format in Scila, used both when storing transactional and for communication between the Server and Client, is built upon the standardized JSON format, see http://www.json.org/.

The messages are written in text format, one message per line. All messages must contain the following data:



This chapter contains a subset of the Scila Message Format Documentation. The messages of interest for this project are, the previously explained, OrderBook, Order and Trade.

5.2.1 Data Types

This chapter contains a specification of the different data types that may be used as values in the message.

Scila Type	JSON Type	Description
String	String	A string is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes.
Long	Number	A 64-bit signed two's complement integer.
Integer	Number	A 32-bit signed two's complement integer.
Float	Number	A single-precision 32-bit IEEE 754 floating point.
Double	Number	A double-precision 64-bit IEEE 754 floating point.
Boolean	Boolean	*true* or *false*
Volume	Number	64-bit signed two's complement integer. Real value * 1000000
Price	Number	64-bit signed two's complement integer. Real value * 1000000
Turnover	Number	64-bit signed two's complement integer. Real value * 1000000
Enum	String	A string representing the enumeration value. Must be one of the specified enumeration values.
Array[Type]	Array	An array is an ordered collection of values. An array begins with [(left bracket) and ends with] (right bracket). Values are separated by , (comma).
Timestamp	Number	64-bit signed two's complement integer. The difference, measured in milliseconds between the message and midnight, January 1, 1970 UTC (if nothing else is indicated).
TsOffset	Number	A number between 0 and 999999 to describe the offset (in nanoseconds) to the millisecond timestamp to make it a nanosecond timestamp
NanoTimestamp	Object	A sub-message with two fields:

Field	JSON Name	Type	Description
timestamp	1	Timestamp	The timestamp.
offset	2	TsOffset	The timestamp offset

Hierarchy	String	An arbitrary hierarchy, used to sort things into a tree structure. Each level is separated with a / (slash). I.e. "/OMX/Equities"

5.2.2 Message Header

The Message Header consists of Message Metadata and information from the Message Source.

Field	JSON Name	Type	Description
Message Type	1	Enum	See "Message Types"
Source ID	3	String	The source ID for the message, see chapter Source ID
Source Counter	4	Long	The source counter. The message number for current day within this Source ID.
Order Book IDs	5	Array[String]	The Order Books IDs for this message, should be empty for Reference Data or if the Message is valid for all Order Books.
Timestamp	6	Timestamp	The difference, measured in milliseconds between the message and midnight, January 1, 1970 UTC
Adaptor ID	7	String	The ID of the Adaptor
Sequence Number	10	Long	The sequence number [Optional]
Scila Timestamp	12	Timestamp	The difference, measured in milliseconds between when Scila Server received the message and midnight, January 1, 1970 UTC [Optional]

5.2.3 Message Types

The internal protocol in Scila is a normalized model of the transaction flow, with a minimum of fields required for the different type of transactions. The messages can be extended to add appropriate fields and functionality required by different customers. The following chapters contain lists of the "core" data fields for different transactions in Scila. Please note that all types may not be relevant to every exchange.

Message	Value	Comment
Barrier In Option	BIO	
Bond	BON	
Barrier Out Option	BOO	
Barrier Warrant	BWA	
Corporate Action	CA	
Reference Price	CST	
Equilibrium Price	EP	
Equity	EQ	
End User	EU	
Future	FU	
Generic Order Book Event	GOE	
Group	GR	Order book group message
GroupChange	GRC	Order Book group change
Index Price	IP	
Index	IX	
MarketMember	MBR	
News	NW	
Option	OP	
Request For Quote	RFQ	
Rights	RI	
Strategy	ST	
MarketUser	USR	
Warrant	WA	
Exchange Rate	XR	
Order Event	1	
Trade Event	2	
State Change	4	
Heart Beat	5	
Order Book Data	6	
Tick Size Table	7	
Instrument Data	9	
External Open Interest	11	
External Position Update	12	
External Detailed Position Update	14	
External Trade Update	15	

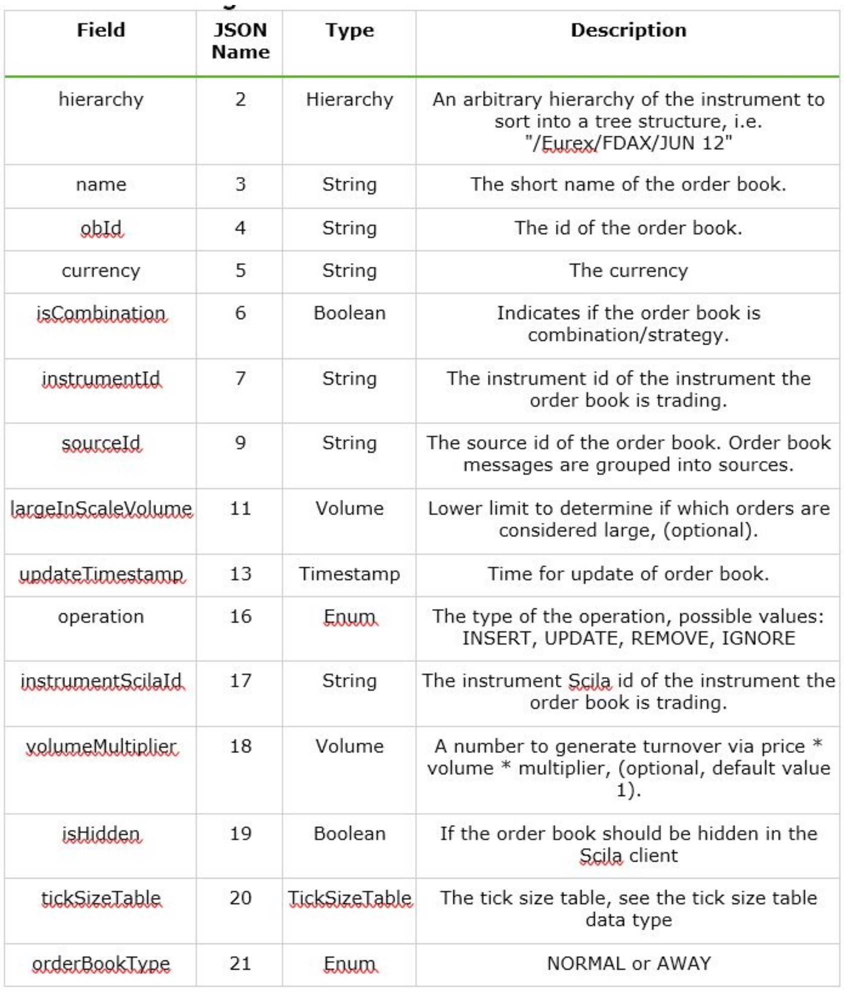
5.2.4 Order Event Message

Field	JSON Name	Type	Description
obId	2	String	The id of the order book. Some kind of instrument identifier.
timestamp	3	Timestamp	The timestamp of the order event (System timestamp)
endUserRef	4	String	The end user that owns the order (if available)
member	5	String	The member that owns the order
user	6	String	The user/trader that owns the order
volume	7	Volume	The total volume of the order
orderId	8	String	The ID of the order. Unique per order book.
price	9	Price	The price of the order
subPrio1	10	Long	A value to determine priority of the order in the order book. Secondary sort criteria, a lower value is always better. Normally this criteria is used for different special order types like AoN, dark orders etc. (Only required if applicable)
subPrio2	11	Long	A value to determine priority of the order in the order book. Third sorting criteria, a lower value is always better. Normally a time stamp or sequence number is used for this purpose.
buy/sell	12	Boolean	True if buy order, false if sell.
operation	13	Enum	The type of operation: INSERT/UPDATE/CANCEL/REPLACE
sourceOfEvent	14	Enum	SYSTEM/USER
publicVolume	15	Volume	The order's public volume
isValidForTrading	16	Boolean	Indication if the order is available for matching.
minimumVolume	17	Volume	For orders with a condition on a minimum volume for execution (optional).
dark	18	Boolean	Flag for dark orders (optional)
obShortName	20	String	The short name of the order book.
quote	21	Boolean	Flag for quotes (optional)
ownerType	22	Enum	The owner type of the order: CUSTOMER/PRINCIPAL/ MARKET_MAKER (optional)
triggerPrice	23	Price	Trigger price for stop orders (optional)
triggerCondition	24	Enum	Trigger condition for stop orders (ASK_OR_HIGHER, ASK_OR_LOWER, BID_OR_HIGHER, BID_OR_LOWER, LAST_PAID_OR_HIGHER, LAST_PAID_OR_LOWER, SHORT_SELL). (optional)
originalPublicVolume	25	Volume	The original public volume of the order.
previousOrderId	26	String	The order id of the previous order in case of update/replace. Used to link together orders that change order ID when updated (If not available, the orderId of this order will be used for this purpose)
isImplied	27	Boolean	Set to true if the order is implied (optional)
timestampOffset	28	TsOffset	The timestamp offset.
endUserId	30	String	The end user ID (optional)
marketDataType	35	Enum	Set if the order event originates from market data (MARKET_BY_LEVEL, MARKET_BY_ORDER) (optional)
marketDataDuplicate	36	Boolean	In case of a market data order, set this value to true if the order has MARKET_BY_ORDER resolution and the order is a duplicate of an order existing in the private order book (optional)

5.2.5 Trade Event Message

Field	JSON Name	Type	Description
askEndUserRef	1	String	The ask end user reference (if available)
askMember	2	String	The ask member of the trade
askUser	3	String	The ask user/trader of the trade
bidEndUserRef	4	String	The bid end user reference (if available)
bidMember	5	String	The bid member of the trade
bidUser	6	String	The bid user/trader of the trade
price	7	Price	The price of the trade
volume	8	Volume	The traded volume
obId	9	String	The id of the order book. Some kind of instrument identifier.
timestamp	10	Timestamp	The timestamp of the trade event (System timestamp)
tradeId	11	String	The unique ID of the trade.
timeOfTrade	12	Timestamp	The timestamp when the trade occurred.
typeOfTrade	13	Enum	The type of the trade event: (NEW/CANCEL/ UPDATED/TRADE_HALF/ TRADE_HALF_CANCELLED/ TRADE_HALF_UPDATED/ EXTERNAL_MARKET)
bidSideAggressor	16	Boolean	A flag for if the bid side is aggressor or not
askSideAggressor	17	Boolean	A flag for if the ask side is aggressor or not
obShortName	20	String	The short name of the order book.
updateLastPaid	21	Boolean	A flag for if the trade should update the last paid statistics (optional)
updateHighLow	22	Boolean	A flag for if the trade should update the high/low statistics (optional)
updateVolumeTurrnover	23	Boolean	A flag for if the trade should update the volume/turnover statistics (optional)
subTypeOfTrade	24	Enum	The type of the trade: (AUTOMATCH, TRADE_REPORT, AUCTION, MARKET_DATA_TRADE)
bidSideAccount	26	String	The bid side account (optional). Used for position keeping.
askSideAccount	27	String	The ask side account (optional). Used for position keeping.
bidSideOrderId	28	String	The order ID of the bid side
askSideOrderId	29	String	The order ID of the ask side
timestampOffset	33	TsOffset	The timestamp offset
timeOfTradeOffset	34	TsOffset	The timestamp offset for the time of trade (optional)
bidEndUserId	36	String	The bid side end user ID (optional)
askEndUserId	37	String	The ask side end user ID (optional)
bidSideOwnerType	38	Enum	The owner type of the bid side order: CUSTOMER/PRINCIPAL/ MARKET_MAKER (optional)
askSideOwnerType	39	Enum	The owner type of the ask side order: CUSTOMER/PRINCIPAL/ MARKET_MAKER (optional)

5.2.6 Order Book Message



Field	JSON Name	Type	Description
hierarchy	2	Hierarchy	An arbitrary hierarchy of the instrument to sort into a tree structure, i.e. "/Eurex/FDAX/JUN 12"
name	3	String	The short name of the order book.
obId	4	String	The id of the order book.
currency	5	String	The currency
isCombination	6	Boolean	Indicates if the order book is combination/strategy.
instrumentId	7	String	The instrument id of the instrument the order book is trading.
sourceId	9	String	The source id of the order book. Order book messages are grouped into sources.
largeInScaleVolume	11	Volume	Lower limit to determine if which orders are considered large, (optional).
updateTimestamp	13	Timestamp	Time for update of order book.
operation	16	Enum	The type of the operation, possible values: INSERT, UPDATE, REMOVE, IGNORE
instrumentScilaId	17	String	The instrument Scila id of the instrument the order book is trading.
volumeMultiplier	18	Volume	A number to generate turnover via price * volume * multiplier, (optional, default value 1).
isHidden	19	Boolean	If the order book should be hidden in the Scila client
tickSizeTable	20	TickSizeTable	The tick size table, see the tick size table data type
orderBookType	21	Enum	NORMAL or AWAY

6 Deliverables

For each deliverable step Scila wants an executable system that we can run on our internal and cloud test servers.

6.1 Component 1

Create an application using Apache Spark which is able to read the Scila TX files. Use the Apache Spark SQL module to query the data. This component is the core of the solution and should be used by the other components.

Answer the questions:

1. Using the Scila tx-format (events sorted in time per instrument group). How do we take advantage of the existing format of trade data for optimum performance when integrating with Apache Spark?
2. How can the data be partitioned to provide highest possible performance for simple SQL queries?

6.2 Component 2

Using Component 1 to query data and implement alert rule which finds cases of market abuse of the type: Spoofing.

The definition of spoofing is quite wide. It merely means that a market participant is putting orders into the market without intent to trade (see: https://en.wikipedia.org/wiki/Spoofing_(finance)).

Here we'll analyze one particular scenario, to find occurrences where a participant has tried to manipulate the market prices by entering and cancelling large orders on the opposite side just before a trade.

6.2.1 Spoofing Alert Rule Requirement Specification

Scenario

A participant intends to trade on one side of the market. To execute his order at a better price he first enters a large order on the other side of the market, hoping that other participants will improve his price or add more volume. He then cancels his spoofing order and enters an order on the other side and gets a better price than he would have otherwise.

Parameters

Minimum Spoofing Order Value <MinSpoof>
The minimum value of the spoofing order

- Spoofing Order Cancellation Percentage <SpoofCxlPerc>
  The minimum part of the spoofing order that needs to be cancelled
- Spoofing Time <SpoofTime>
  The time, before the trade, that spoofing orders are looked for.
- Participant level
  The level of the participant. The levels are defined in a hierarchy where 'member' is the top level, 'user' the second and 'endUserRef' the third.

Implementation suggestion

1. Find all trades that were executed during continuous trading (SubTypeOfTrade == AUTOMATCH).
2. Find all orders with the same participant as the trade on the opposite side within <spoofTime> prior to the trade that are of <minSpoof> value or more. (Note that this check will be for both bid and ask side.)
3. For the orders that fulfill 2), check that at least <spoofCxlPerc> is cancelled within <spoofTime>
4. Alert for each fulfilled scenario

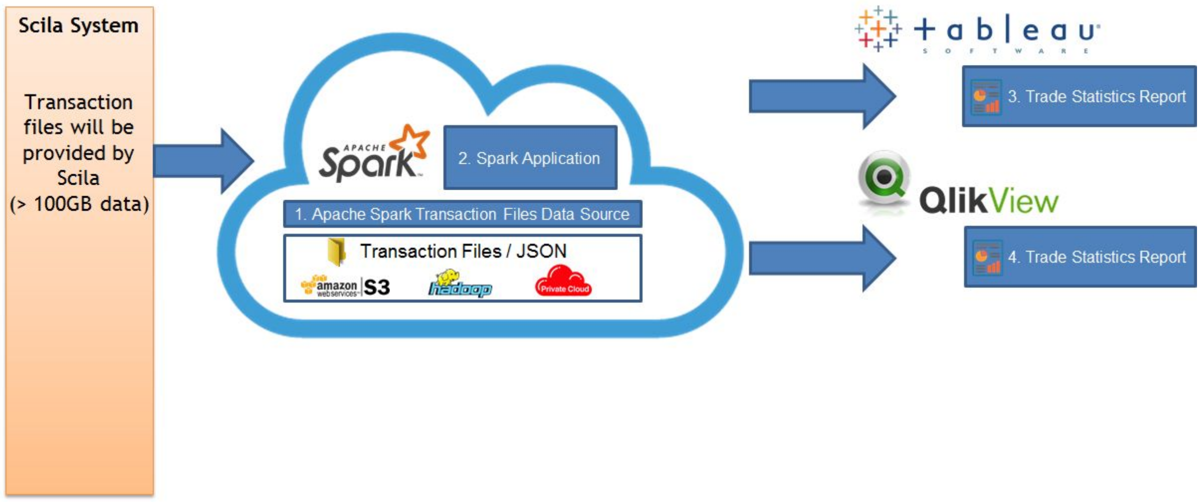
6.3 Component 3

Use Component 1 and Spark Machine Learning Library (MLLib) for Anomaly Detection. Find outliers, using unsupervised learning (like clustering), in Price and Volume or Participant, of Orders that deviates so much from other observations to arouse suspicion.

https://spark.apache.org/docs/latest/ml-guide.html

6.4 Component 4 - (extra task, if time allows)

Integrate Component 1 with QlikView and/or Tableau to create drill down reports for statistics per participants and instrument.


Scila System
Transaction files will be provided by Scila (> 100GB data)
Spark
2. Spark Application
1. Apache Spark Transaction Files Data Source
Transaction Files / JSON
amazon web services S3
hadoop
Private Cloud
tableau SOFTWARE
3. Trade Statistics Report
QlikView
4. Trade Statistics Report

6.4.1 Order and Trade Report Specification

Create a report in Tableau or QlikView with the fields listed below. The report shall show one row per instrument. The user shall be able to specify the time than the reports span.

Field Name	**Description**
Instrument Name	Name of instrument
Date Time Span	Date-time span of report parameter
VWAP	Volume Weighted Average Price
Number of Trades	Number of trades
Trade Volume	Sum of volume on all trades

Turnover	Sum of volume times price on all trades
Number of Orders	Number of orders
Number of Ask Orders	Number of orders on ask side
Number of Bid Orders	Number of orders on bid side
Order to Trade Ratio	Number of orders per trade
High Price	Highest price
Low Price	Lowest price

7 Implementation

7.1 Tools and Languages

7.1.1 Required

Java 8

Apache spark: https://spark.apache.org/

7.1.2 Recommended Integrated Development Environments

IDE: IntelliJ IDEA https://www.jetbrains.com/idea or Eclipse https://eclipse.org

Maven for building: https://maven.apache.org

7.1.3 Optional but recommended

Spring Framework: https://spring.io

Spring Boot: https://start.spring.io/

7.1.4 Communication

For questions and discussions with Scila there will be a Google Group available.

Email: projectcs@scila.se

7.2 Non functional requirements

1. The solution must be able to consistently analyze billions of rows of data without generating errors and with response times on the order of 10s or 100s of seconds.

2. The solution needs to deliver interactive performance on known query patterns i.e. return results in no greater than several seconds on small data sets (on the order of thousands or millions of rows).

8 References

1. Order Book, Investopedia
   http://www.investopedia.com/terms/o/order-book.asp
2. Order, Wikipedia
   https://en.wikipedia.org/wiki/Order_(exchange)
3. Trade, Wikipedia
   https://en.wikipedia.org/wiki/Trade_(financial_instrument)