eXchains

CS4160 Blockchain Engineering (2017/18 Q2)

Project Report by:
Joseph Verburg 4018575
Michał Łoin 4587324
Niels Hokke 4610148
Jetse Brouwer 4615964
Evgenia Domnenkova 4625633

1. Introduction to Imbalance Market

The energy market consists of multiple markets with the main goal of balancing production and consumption. Each market works on a different timescale with different terms. The 4 most important markets are the Long-term market, day-ahead market, intra-day market and the unbalance market.

The imbalance market is the most volatile market. This market is led by the Transmission system operator (TSO, TenneT in the Netherlands) and serves the sole purpose of maintaining a stable frequency of the grid. When an imbalance arises TenneT has multiple reserves at its disposal. These reserves consist of parties who have the ability to regulate their consumption or production up or down relatively fast (depending on which reserve group they're in). The primary reserves will be used to solve immediate imbalance. The secondary reserve is the market that is used if the grid hasn't stabilized after the 15 minutes of using the primary reserves. At last there is a tertiary reserve, also called the emergency pool, for long term high-power regulation power.

2. Problem Statement

With consumers of electricity now also becoming producers at a higher rate, more imbalance is created on the net. To maintain security of supply the imbalance should be kept within the limits of the grid. Currently the transmission system operator (TSO: TenneT in Netherlands) is responsible for maintaining this balance. A problem arises when the production becomes more and more decentralized, and the balancing is still the duty of a centralized body, it becomes harder to respond on the highly fluctuating consumption and production of the different actors.

The aim of this project is to design and prototype a decentralized market and a so called 'smart grid' where participants of the smart grid can dynamically match their supply and demand to minimize imbalances.

3. General Approach to Balancing

The system should be able to be deployed in co-operation with the current electrical grid as it unrealistic to expect major changes in the grid topology in the foreseeable future. The grid is currently laid out in a manner where several households (1 or more streets) are clustered together sharing a physical cable. These clustered cables join at small brickhouse-substations spread out through the city. These substations are clustered together on a 10KV network, where one cluster roughly serves a city. Between nearby cities a 50KV net grid is clustering those together, from there they are clustered on a 110 -150 KV grid and again at a 380KV grid (which also connects with neighbouring countries). In the current situation, the grid is balanced by the TSO, who buys capacity to up- or downregulate the consumption or production at companies spread out over several levels of the grid, and does so when required.

For balancing the grid, matching consumption and production at the highest level is not enough. Even when nationwide balance is achieved overload can happen due to unreasonably large imbalances on lower levels. Therefor we believe that balancing should, in the first place, happen within same cluster (e.g. a single street) before propagating it up to a higher level (e.g. district or city). When looking at an example of at the lowest, the spikey behaviour of a single household is smoothened out by combining the households per cluster, and the smart grid aims to spread out the load equally over time. The remaining imbalance is then propagated to a higher level where bigger players (with more capacity but lower flexibility) can partake in the balancing game. For every step to a higher level the peaks become smoother and more predictable, which is very desirable for the big powerplants that will have to supply the energy in moments of energy shortage.

4. Balancing Using Blockchain

To enable the actual balancing between the participants of the smart grid (which will be called clients from now on) a platform must be offered to gain knowledge of the current state, the expected state and to reach agreements on energy exchanges. For this project we will explore the possibilities of using a blockchain for providing such a platform.

As discussed earlier, the focus should first lie on balancing within a single cluster, therefor there is no need for connecting every client to every other client. This creates the possibility of creating a blockchain per cluster instead of having to design a blockchain for all households in the system, which reduces the amount of transactions per second per blockchain significantly.

The knowledge per cluster will then be aggregated and propagated as a single entry up to a higher level blockchain, where bigger parties can join the energy balancing market. The aggregation results in individual information being lost when propagating up, which reduces the amount of data being send around but does not create problems as this information is not needed in the higher levels.

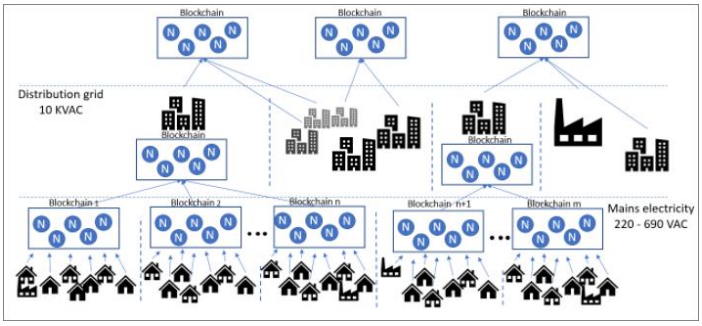


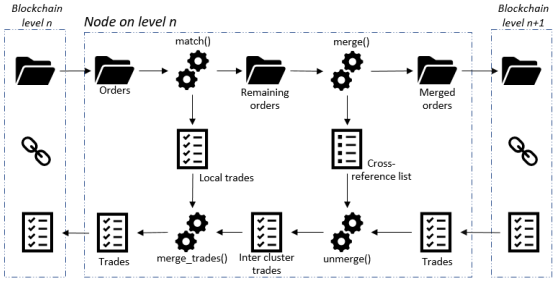
Figure 1: A possible data structure with a blockchain per cluster.

By following the physical lay-out for clustering the nodes we end up with a structure that could be visualized as a tree structure where the dataflows from node to node are only through their parent. A figure depicting such a tree is given in Figure 1. Here the circles containing the letter 'N' are the nodes that host the blockchain, the black symbols depict the clients (which can be households, industries and companies specialized in balancing).

5. Balancing Algorithm

The main idea behind the balancing algorithm is that users (clients, aka smart meters and intelligent household agents behind them) submit to the blockchain their expected consumption, production and the flexibility they can afford/offer for the coming balancing timeslot. The flexibility is the indication from the client to what extend they are ready to increase/decrease their production/consumption and at which price. The more flexibility there is on offer, the easier it is to balance the grid. The prices given in the flexibility data are not fixed prices but limits similar to exchange markets. 'Ask' price is the maximum price the client is willing to pay for that amount of energy, where the 'Bid' price is the minimum the client wants to sell that amount of their energy for.

As the model is currently expected to be compatible with the existing grid, no user is expected to be denied the requested energy within the predicted consumption. Therefore, every user, before being able to join the balancing, is required to have a contract with a back-up supplier of the grid. The back-up supplier is introduced into the model for the worst-case scenarios when the whole grid cannot balance itself. The idea is that the back-up will be used as little as possible but needs to be there in case of emergency.



When it comes to matching supply ('bid') and demand ('ask'), the buyer almost always pays less than their maximum, and the seller almost always makes more than their minimum. To encourage clients to place their bids at their personal extremes, the algorithm first matches highest buyer with lowest seller (just as the stock market), as those orders generate overlap and

thereby enable more matching, and consequently help to balance the grid. The matching first happens at the cluster level. Then the consolidated imbalance is sent up to the next cluster, and the same matching algorithm is executed at the next level.

6. Considerations

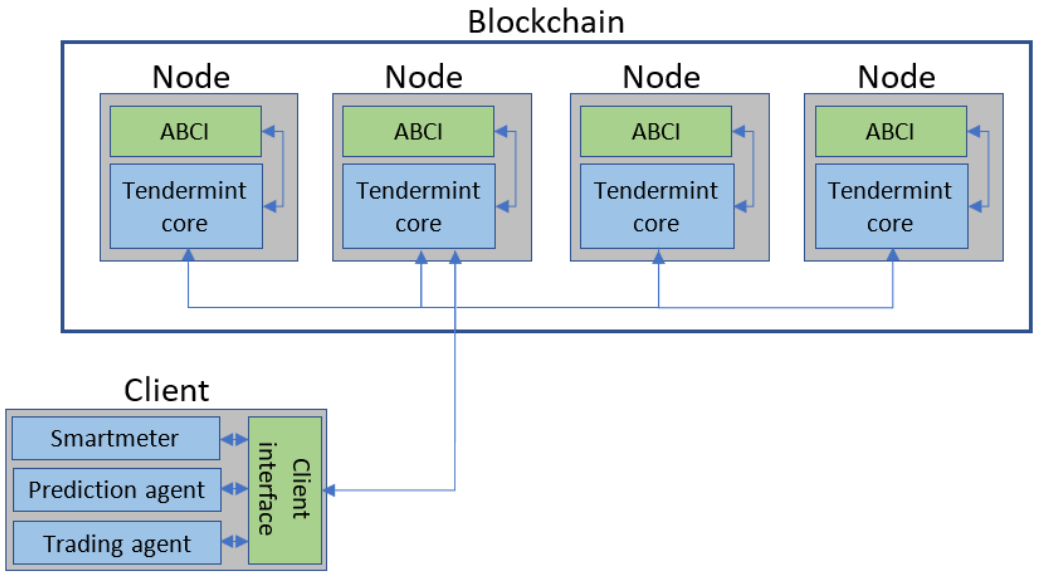
As part of the project, the team familiarised themselves with the following material and information:

- Princeton blockchain lectures
- Blockchain market research
- Blocklab paper
- Lecture notes Distributed Algorithms
- Master thesis of SjorsHijgenaar
- Satoshi paper
- Tendermint and PoS problems
- Tendermint
- Graph illustrating the consensus mechanism of Tenderminder
- Tendermint thesis paper
- Blockchain as a service system based on Tendermint
- Cosmos.Network
- Etherium
- Casper
- Hyperledger-fabric
- Ripple
- Corda
- Iota
- Waves
- Bitshare

7. Technical Prototype Overview

Layers:

Client: Smart-meter transactions simulated by a python script with help of a static csv file for data simulation
Blockchain: Tendermint Core and Custom ABCI written in python + Validator VMs in Docker
Visualisation: Thin client written in Node.js to visualise blockchain process



Blockchain and Tendermint

The decision to use blockchain technology is based on the distributed nature of it. For a creating a decentralized market it is by definition impossible to have a central place for data storage to keep track of the orders and trades while remaining decentralized. To ensure that all the data is consistent amongst the partakers and is stored in an immutable manner a technology must be selected for this; blockchain suits this task perfectly. It furthermore enables to have multiple parties to run a single market without out having to explicitly trust each other as they all contribute to, and benefit from a trustworthy and stable infrastructure.

With the energy market already being a highly regulated market. As a consumer it is not possible to get a connection without the connection being installed by a certified technician, and as a company one has to buy himself in and must meet certain criteria before being allowed on the market. Because of these regulation, a permissioned blockchain seems

the best choice as this mirrors the real-world aspect of joining the energy market. For choosing between proof of work or proof of stake the decision comes naturally to choose for proof of stake as due to the permissioned and on the fact that solving an energy related problem by doing energy intense calculations is counter-effective.

Tendermint was chosen as it is a permissioned, BFT, proof of stake consensus algorithm and therefore matches well with the demands for this project. Tendermint offers a lot of flexibility as it is written in a modular approach, where the Tendermint is agnostic of the blockchain application logic. The use of sockets for communication between the Tendermint core and the ABCI (Application-blockchain interface) enables the use of a wide variety of programming languages for writing the ABCI. We also contacted Sjors Hijgenaar, who uses Tendermint within CGI for a energy related problem and he promised us technical support if there would be any unsolvable issues. Due to the familiarity of python among the project member python was chosen for the ABCI.

Technical challenges of Tendermint

While designing and implementing our solution several technical difficulties had to be faced. One of the problems originates from Tendermint being an immature software project is the lack of technical documentation and community to rely on for support. Another issue that needed resolving was that the examples in python were outdated and no longer working, updating them to a newer version was 'left as an exercise to the reader'. Furthermore, there were substantial bugs in the earlier version of the Tendermint core which were fixed during the project, however without backwards compatibility which resulted in reworking several parts of the ABCI. The feature to identify the initial proposer of a block is still pending on the request list, however as we relied on this information a workaround had to be implemented.

Tendermint and Balancing

To facilitate the blockchain, Tendermint is used. Tendermint consists of two main technical components: the Tendermint core and the 'Application BlockChain Interface' (ABCI). The Tendermint core is the consensus layer responsible for receiving, verifying and reaching consensus on transactions and blocks. The ABCI runs on 'top' of the core consensus layer and supplies the developer with an interface to interact with the blockchain. Every time a client wants to place information on the blockchain a transaction is send to the Tendermint core. The core will call checkTX() in the ABCI to verify the validity of the transaction (based on the state of the blockchain and other factors that are defined within checkTX() by the developers). When the receiving node considers it a valid transaction it will add it to its pool of transactions and will gossip it to other nodes that, in their turn, will verify it as well and will add it to their pool of transactions. The Tendermint core manages the block creation rounds: every round a node is elected to propose a block to be added to the chain. After a round of verifying and voting the block will be added to the chain or rejected.

On top of block creation rounds that are managed by the core, the ABCI of this project needs to manage balancing rounds. As soon as a new balancing round is announced the nodes start a timer for 10 minutes. In these 10 minutes clients can report their usages to the blockchain. After a node's timer expires it submits a 'Begin Balancing' transaction. Other nodes receive this transaction and will verify it with their own timing. With enough agreement that the timing is right, it will be added into the blockchain. At this point both the nodes and the clients are aware that no more transactions will be accepted, and the nodes have 5 minutes to balance the trades and send a transaction up to the next level blockchain.

All the nodes run the balancing algorithm defined within the ABCI, but only one is allowed to submit balancing transactions to the blockchain. All the other nodes verify that the transactions posted by the chosen node are in line with their own results. Since the balancing is designed to be fully deterministic a certain unique input will always result in the exact same unique outcome, and therefore mistakes or malicious intent can be easily detected. The node that is allowed to submit the balancing transactions is picked based on a random number generator seeded with the hash of the previous block to ensure unpredictable but fair and deterministic chances of being the next balancer.

8. Conclusions and Next Steps

Project Conclusions:

- The working prototype demonstrates the core ideas, that the team developed about the approach, are technically feasible
- Tendermint's idea about the BFT Consensus Core and Custom ABCI is a good concept, however Tendermint as a platform needs to mature to make it easier to develop and deploy on it

Possible Next Steps:

- Enhance ABCI from prototype to a more deployable solution, including future enhancements of Tendermint that may make implementation simpler
- Enhance balancing algorithm logic
- Implement smart contract and billing features
- Implement intelligent agents running on the smart meters
- Test the solution on physical infrastructure (e.g. real smarts meters, real households, real validators from the balancing community, etc)
- Pilot the process/technology with real world parties (e.g. households, energy companies, Tennet, etc)

Appendix A: Energy Balancing Algorithm

The main idea behind the balancing algorithm is that users place their expected consumption and production for the coming timeslot on the blockchain. But while posting the expectation may give an insight on the coming time slot, it does not solve any imbalances. To enable balancing the users also have to inform on how flexible they can be with their production and consumption. This is done by posting the flexibility for both the production and consumption, stating how much energy he is willing to buy or sell and for what price. Selling energy can be by both increasing the production or decreasing the consumption and buying by doing the inverse. (E.g. postponing the charging of the client's e-vehicle or even discharging it for a financial compensation)

It may be the case that the consumption and production cannot be perfectly matched. As the project description states that the solution should be compatible with the current grid, it would be impractical and unreasonable to deny the user the requested energy. To overcome this issue the notion of a back-up supplier is introduced.

This back-up supplier reaches an agreement with the user on a price at which it will fill in any remaining energy gap. This agreement must is made before the client will participate the system.

By doing so the system can be a drop in replacement in the current electrical grid, if a user does not wish to partake in the balancing he can just post his consumption and production and keep is flexibility fixed at zero, and the back-up supplier will act as a normal utility company.

Clients report

Per round of balancing all the clients submit their data to the blockchain using the app.DeliverTx(tx) (where the argument 'tx' is a bytearray Protobuffer representation of the data, example data is displayed below). The data consists of the consumption and the production of the previous x minutes. Furthermore, it sends the prediction of the production and consumption for the following timeslots and the flexibility it has to offer for the coming time slots. Since the price per offered energy unit may not be constant (A client may offer the first x kwh for $0,50, but can offer an additional y kwh only if the reimbursement will be higher), the client can send its price curve as a piecewise linear function as the pair '€/KWh' : 'Wh'. A negative volume means the consumer is willing to decrease the corresponding consumption or production. A positive volume indicates increasing the consumption or production.

```
'MessageType' : ClientReport,

Data : { 'UUID'    : 0010238408963,
         'Timestamp' : 123768238479,
         'Consumption'      : 123.45 Wh,
         'Production'       : 45.321 Wh,
         'Def. c price'     : 123.45 Wh,
         'Def. p price'     : 45.321 Wh
         'PredictedCons'    : {'t+x' : 232.43 Wh, 't+2x' : 352.87,  't+3x' : 32.54 Wh ...  't+y-x' : 623.43 Wh},
          'PredictedProd'   : {'t+x' : 4.433 Wh, 't+2x' : 0,  't+3x' : 32.54 Wh ...  't+y-x' : 74.43 Wh},
          'consFlexibility' : {'€0,12' : 110 Wh, '€0,24' : '-55 Wh, '0,85'' : 20 Wh},
          'prodFlexibility' : {'€0,12' : -12 Wh, '€0,24' : ' 55 Wh, '0,85'' : -21 Wh},
       }
```

Guaranteed consumption and production

The amount of energy stated as 'predicted consumption' is guaranteed to be delivered. This can be seen as the minimum amount of energy that the user requires. If this demand cannot be filled in by offers from other clients, the back-up supplier will sell the remaining energy for a pre-determined price to the client.

The amount of energy stated as 'Predicted Production' is guaranteed to be accepted on the net. If it can't be fully matched, the back-up supplier will buy the remaining energy for a pre-determined price.

In the current situation the utility companies do not have separate prices or placed any limits on the client's production. In the future however, it seems possible that clients will reach an agreement on forehand on how much they are allowed to produce and at what price.

Consumption and production flexibility

'Consumption flexibility' and 'production flexibility' is the amount of flexibility the user is willing to deploy send in the form a list.

When 'Consumption flexibility' is positive or 'production flexibility' is negative, it means the user is willing to increase its consumption or decrease the production by at most the given amount of energy. Both can be seen as an 'Ask' for energy from the grid (as it is irrelevant and technically impossible for the grid to distinguish an increase in consumption or decrease in production).

(however the sum of all the negative consumption can never be greater than 'PredictedCons)'

When 'Consumption flexibility' is negative or 'production flexibility' is positive, the user is willing to decrease its predicted consumption or increase the predicted production by at most the given amount of energy. Both can be seen as 'Bid' of energy to the grid

The prices given in the flexibility are not fixed prices but limits similar to exchange markets; for 'Ask' it's a maximum price the client is willing to pay for that energy, where the 'Bid' price is the minimum the client wants to receive for that amount of energy. This increases the chance on matching as it is quite unlikely two parties offer for the exact same price.

Matching 'Asks' and 'Bids'

Since the ask price is a maximum and bid price is a minimum, settling for the arithmetic mean when matching is beneficial and fair for both parties. The buyer always pays less than its maximum, and the seller will always make more than the minimum.

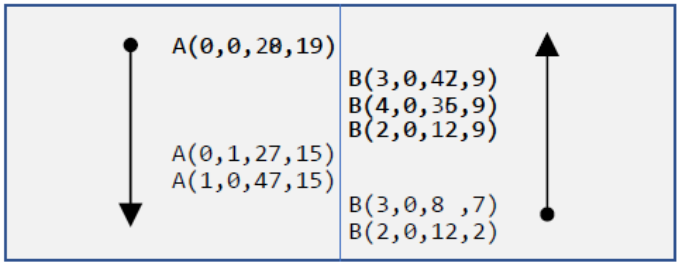
A bigger overlap between the highest ask price and the lowest bid price increases the chance of matching. To encourage clients to place their order at their personal extremes, we first match highest buyer with lowest seller (just as the stock market) as those orders generate overlap and therefore enable more matching. In this way people generated flexibility are rewarded with a higher chance of matching before running out before counter orders and potentially making/saving money.

Furthermore, this client is rewarded with potentially better prices; as the highest buyer will matched with lowest seller, he might pay less than the second highest buyer (e.g. buy 9 + sell 1 = 5, buy 8 + sell 4 = 5,5).

To perform the balancing the system tries to match as much as asks and bids as possible, similar to how the stock market works.

It does so by first creating a list of orders that overlap (i.e. bids that are lower than the highest ask, and asks that are higher than the lowest bid). Any orders not in these lists cannot be filled in by definition. The remaining order list will sorted first on price and then on volume, an example of such an order book can be seen in figure XXX.

Order book:



Transactions

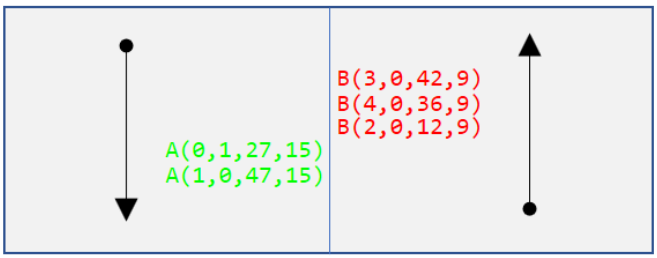
Ask	=	a(uuid, ordered, volume, price)
Bid	=	b(uuid, ordered, volume, price)
Trade	=	t(uuid, order, volume, price)

From there one it will start matching with the highest buy price and the lowest sell price. If there are multiple orders at the same price they will be grouped. At first the highest buy and the lowest match will be made resulting in a trade of a volume of 12 (since that is all the bid could deliver) and added as a trade for both the bid order and the ask order, in this way every client knows how much he traded with the counterpart remaining anonymous. After this trade still 8 of the 20 is open and will remain in the order book to be matched with the next lowest bid. After performing all the straight forward matching the trade book and order book will look something like figure XXX.

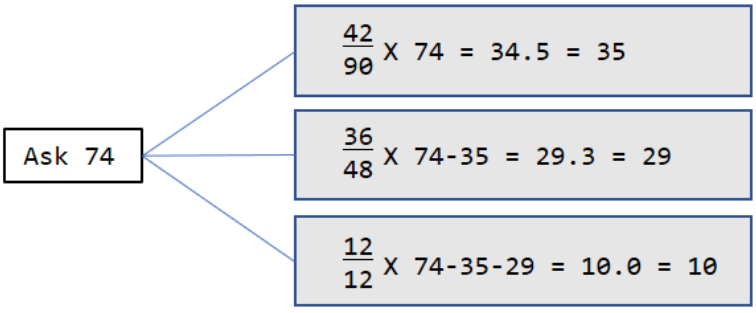
Trades:

```
T(2,0,Bid,12,11)
T(0,0,Ask,12,11)
T(3,0,Bid, 8,13)
T(0,0,Bid, 8,13)
T(0,1,Ask,27,12)
T(1,0,Ask,47,12)
```

Order book:



The next orders in the order book are of the same price and therefore are equally responsible for generating overlap. To treat them fair all order of the same price are combined and treated as a single order. The volumes of the combined ask and the combined bids are compared, and the smallest will be proportionally spread out amongst the bigger one. In this case algorithm spreads out 74 volumes over 90 volumes. The providers of the bigger volume get their trades fulfilled proportionally to the amount they offered.

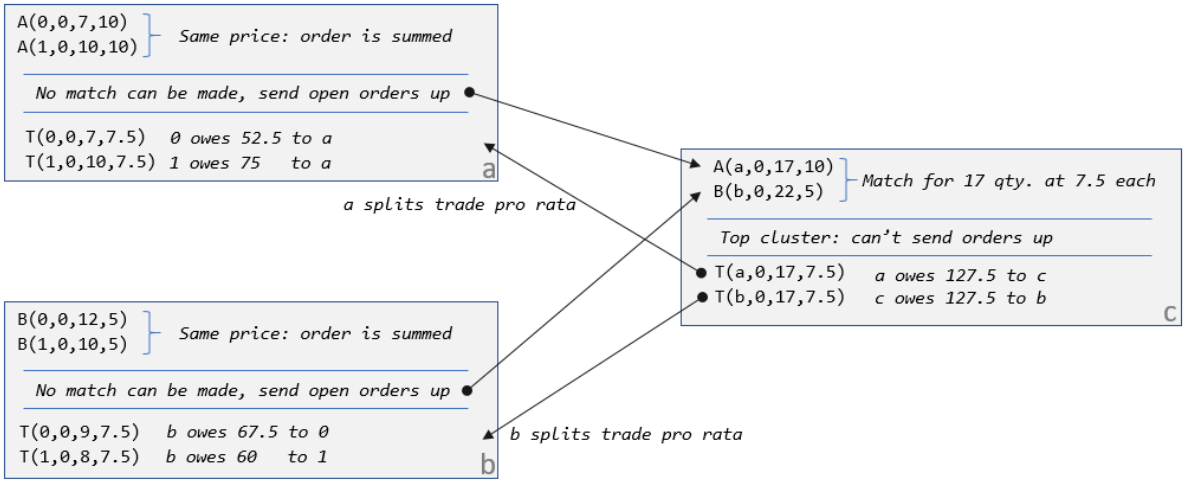


The splitting up is done in steps and rounded in between very step to ensure that no rounding errors end up to have a difference between produced and consumed energy. (e.g. 1/3, 1/3,and1/3 would round down to 0, 0, and 0 which results in ‘0’ being matched on the larger side but ‘1’ on the smaller side. When rounding in between steps it would become: 0, 1, 0).

To ensure the results of matching algorithm is deterministic all of the order are sorted following the same order and executed accordingly: Price, volume, timestamp, uuid, ordered. Since those values are all obtained from the blockchain (meaning consensus has been reached) all nodes agree on these values and ordering will result in the same order, resulting in the same execution amongst all of the nodes.

Inter-cluster trading

As stated earlier one, it is not unlikely that the imbalance will not be solved within a single cluster. Therefore the remaining open orders will be sent upwards to a higher cluster to try to be matched with other clusters. Before sending up, orders of the same price can be combined this minimizes the data that needs to be send upwards. Also all the orders submitted will be submitted using the uuid of the cluster submitting the trade as the clusters at a higher level do not care about which client submitted the data to the lower cluster.



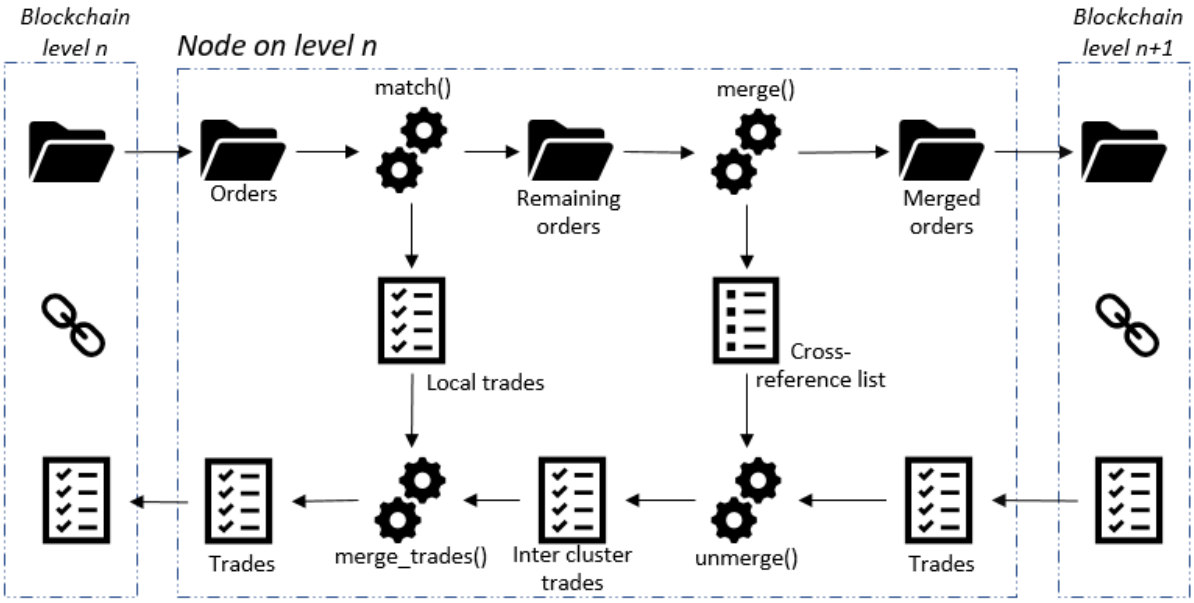
When looking at the example given in fig **XXX**, it can be seen that both cluster have orders that can't be fulfilled locally. They are combined and passed on to the higher level as a single order with the uuid of the cluster instead of the original client's uuid. Since all the orders and trades are labeled per cluster the only need to transfer 2 orders and 2 trades, instead of transferring the 4 original order and the 4 resulting trades.

In the resulting trades in figure **XXX** it can also be seen that every client only ever gets money from or owes money to its own cluster. When, in a later stadium, billing would be introduced these greatly simplifies the payment as every client only has to deal directly with the cluster it belongs to, and every cluster only has to deal with its own clients and at most two other clusters (one below him and/or one above him).

Schematic overview of the algorithm

All of the above mentioned actions done by the algorithm are implemented in 'matchmaker.py'. A schematic overview of all the elements can be seen in figure **XXX**. Here the left and right rectangle represent the blockchain on the level the node runs on, and the level above him. The rectangle in the middle represents the balancing algorithm running on that node. After the bidding round stops, the node collects all the orders in the block chain, and collects them in an orderbook. This orderbook is then passed as an argument to the 'match()' function, which will perform local matching. If there are matches that can be made entries in the trade book are made. Trades that couldn't be (completely) fulfilled are then passed to the 'merging()', which generates an order book to be passed to the next level along with a cross-reference list.

This cross-reference list contains a list of order that were merged in to new order, so when the node gets trades back it knows which orders it originally belonged to. When the higher level could fulfill any of the order, the client will unmerge them to the original order spreads out the trade volume amongst the corresponding order. These new trades are then added to the trade list and will be posted in the blockchain from where the orders came.


Blockchain level n
Node on level n
Blockchain level n+1
match()
merge()
Orders
Remaining orders
Merged orders
Local trades
Cross-reference list
Trades
merge_trades()
Inter cluster trades
unmerge()
Trades