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 his 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.

‘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’

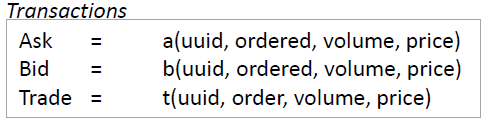
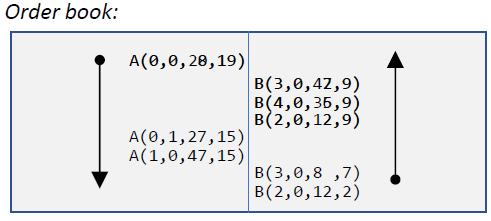
Since the ask price is a maximum and bid price is a minimum, settling for the arithmetic mean when matching is beneficial and fair for both parties. The buyer always pays less than its maximum, and the seller will always make more than the minimum.

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.

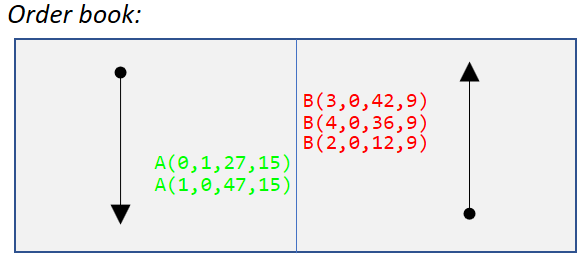
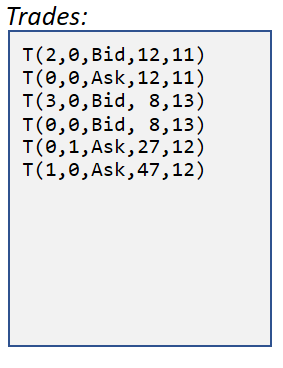
Furthermore, this client is rewarded with potentially better prices; as the highest buyer will matched with lowest seller, he might pay less than the second highest buyer (e.g. buy 9 + sell 1 = 5, buy 8 + sell 4 = 5,5).

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

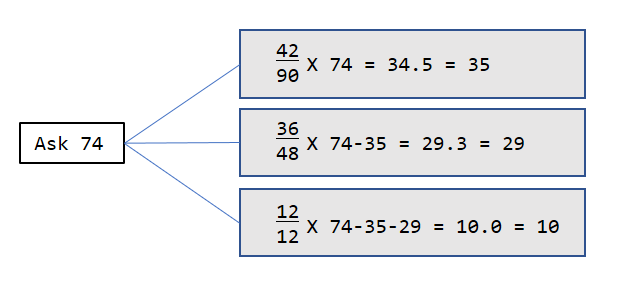
It does so by first creating a list of orders that overlap (i.e. bids that are lower than the highest ask, and asks that are higher than the lowest bid). Any orders not in these lists cannot be filled in by definition. The remaining order list will sorted first on price and then on volume, an example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



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.



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.Tthe 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.

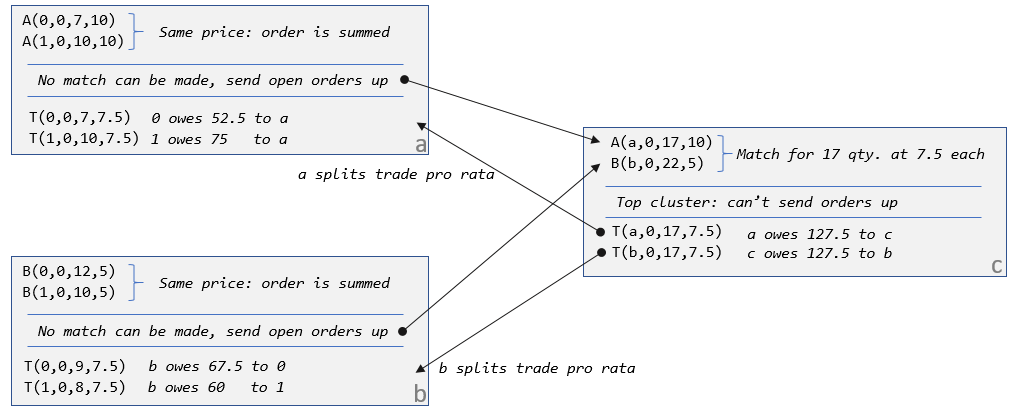


The splitting up is done in steps and rounded in between very step to ensure that no rounding errors end up to have a difference between produced and consumed energy. (e.g. 1/3, 1/3,and1/3 would round down to 0, 0, and 0 which results in ‘0’ being matched on the larger side but ‘1’ on the smaller side.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



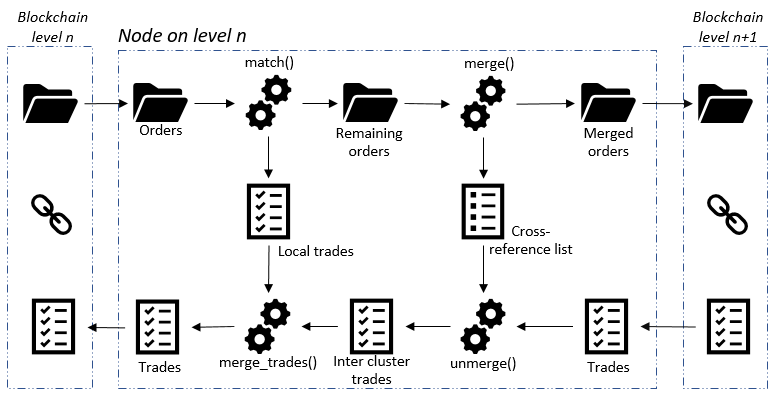
When looking at the example given in fig **XXX**, it can be seen that both cluster have orders that can’t be fulfilled locally. They are combined and passed on to the higher level as a single order with the uuid of the cluster instead of the original client’s uuid. Since all the orders and trades are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



### Presentation structure

<https://www.cpb.nl/sites/default/files/omnidownload/CPB-Beantwoording-vragen-verschillen-Regeerakkoord-en-doorrekening-CPB.pdf> (source decoupling production and consumption price after 2020)

* Introduction:
  + What problem? (imbalance caused by unpredictable consumers and porudercs. consumers becoming producer + intermittent sources vs old fashioned powerplant-household configuration:
  + Why a problem? (introduction in to the grid; there is a physical limit on capacity)
    - Physical grid is already clusters (kind of tree).
    - On the lowest level the imbalances are the smallest, not yet a problem
    - However combined they become large ->overload inter-cluster connections
    - Solve the problem by the root, first try to balance in the cluster, then on 1 level higher, and so on.
* The balancing (easier said than done):
  + Explain notion of backup supplier:
    - Drop-in replacement; can’t cut people off, different price production consumption, maybe even cap the production (very likely in the future; 2020).
  + Clients submit their baseline consumption to keep the household running (all of this by a software agent on their behalf). This is guaranteed; first locally else backup suppliers.
  + Next to that also their flexibility: Don’t really need to charge my car yet, but if it’s cheaper then backup currently might as well do it now. Or might stop charging my car if someone needs it more than me (i.e. willing to pay more for it than it costs me). smart appliances, (fridges, overnight laundry stuff etc.) but also new businesses specialized in storing energy can join and trade energy (cost of wear and tear vs current price etc.)
  + All of this is information enters the ‘energy exchange market’. Loosely based on stock market. Buy price is maximum price, sell is minimum and you settle in the middle so it’s fair for both (always pay less then you want, or get more than you asked). Bigger overlap is bigger chance of match making. That’s why (just as the stock market) we first match highest buyer with lowest seller. (they generate overlap and therefore enable more matching). also because highest buyer is matched with lowest, seller he might pay less than the second highest buyer. (buy 9 + sell 1 = 5, buy 8 + sell 4 = 5,5). Or the second highest buyer might not even match at all.
  + After possible trades are made in a cluster, data is aggregated (minimize data flow) and remaining flexibility is send up, to see if energy can be traded intra-cluster, if nit move up until root.
  + All of this is implemented in a 100% deterministic way so that every unique set of incoming order always generate the exact same set of trades in the exact same order (and no rounding or float errors). Important for consensus later on.
* The blockchain part:
  + Why blockchain:
    - Multiple parties involved in the grid; but not necessarily trust eachother, immutable data; tamper resistant
  + Why tendermint:
    - Proof of stake (proof of work defies the purpose: waste energy to solve energy problems)
    - Permissionsed
    - Agnostic of application but does implement all of the consensus reaching and messageing stuff