CatCoins - Derivative Trading Platform

Fabian Künzler[15−941−842], Michael Nadig[16−918−690], Sandro Padovan[17−721−291], and Christian Birchler[15−924−160]

University of Zurich, Zurich 8006, Switzerland
{fabian.kuenzler,michael.nadig2,sandro.padovan,christian.birchler2}@uzh.ch

Abstract. In this work, we present a decentralized platform to trade derivatives by using smart contracts and blockchains. The platform allows for trading of simplified call and put options on different underlyings in a completely decentralized manner. We present a proof of concept running on the Ethereum blockchain and a web-based graphical user interface.

Keywords: Blockchain · Ethereum · Smart Contracts · Derivatives · Betting

1 Introduction

This introduction gives a short overview of the financial products that can be traded over the newly developed marketplace. Furthermore, the advantages of our solutions and the stakeholder's interests are highlighted.

1.1 What are derivatives?

The goal of derivatives is to hedge yourself against unfavorable outcomes and ultimately contribute to economic growth. But used wrongly and with limited regulation, derivatives can lead financial institutions and the whole financial system into economic turmoil, as we have seen in 2008 with MBS (Mortgage based Securities) and CDS (Credit derivative Swaps) [0]. The ultimate decision of how and if derivatives are useful is an ongoing discussion. Despite being controversial derivatives claimed a trading volume 7 times bigger than the GDP of the entire world in 2016 and is rapidly growing [0].

There exist many different types of derivatives in today's markets. They are used to hedge risk, speculate or to be combined in complex financial products such as structured products. The definition that all of these derivatives have in common, is that the value (from now on called price) of the derivative is dependent on an underlying price [1].

An underlying can be anything with a value. Widely used are underlyings such as stocks or commodities. For the sake of simplicity, we focused on a derivative type called options.

An option gives the person who holds the option the right but not the obligation to buy/sell the predefined underlying for the predefined price, called the

strike price. The variables such as maturity, strike price and the underlying are predefined in the option contract [0]. We focused ourselves on simple European Call and Put options without knock-in or out barriers. Below are the payoffs of these two simple options depicted

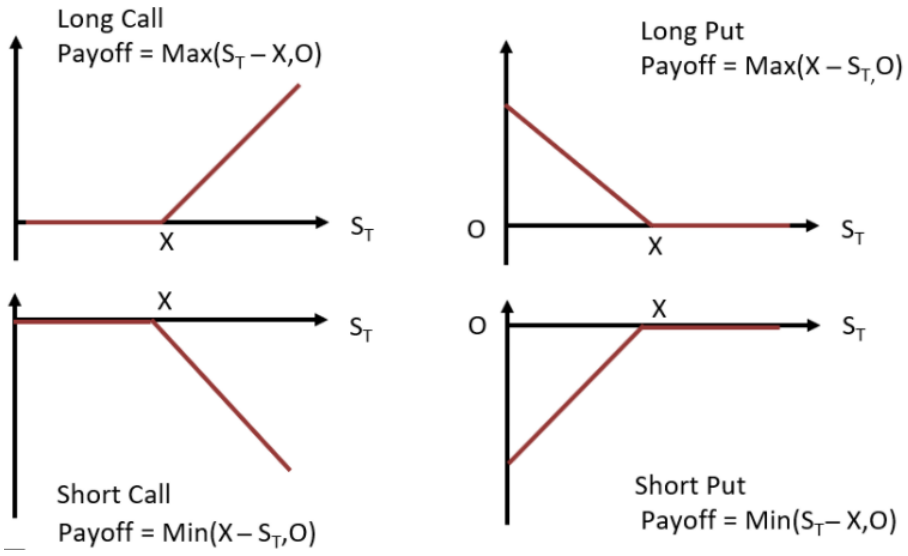

Fig. 1. Payoff of simple Call and Put options

The long position in a Call option is the one who is holding the option and has therefore the right to execute the option at the end of the maturity. If the underlying price (S_t) has risen over the predefined strike price (X) the holder of the Call option executes it. The person who is short delivers the underlying to the holder who can then sell it on the market with a profit. Alternatively, the execution is settled with cash, meaning the short position pays the long position the profit directly [0]. We focus on the latter settlement.

The Long Put on the other hand gives the option holder right but not the obligation to sell an underlying for a predefined price. Therefore the holder hopes for a decreasing price of the underlying to make a profit [0].

We decided to restrict this even further to make the financial aspects simpler. Therefore we restricted the maximum loss of the short call to a predefined value we called the bet. This caped structure can be achieved in the real world by combining a Short Call with a Long Call but with different strike prices. Furthermore, we defined that the winning party wins it all. This means the profit/loss is not linearly increasing or decreasing. Thus, the before mentioned restrictions result in the following payoff diagram.

The nature of these payoffs has also some similarities to future and forward contracts which will not be discussed here.

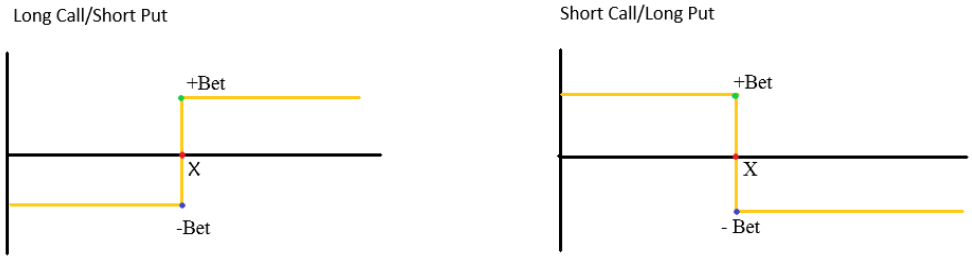

Fig. 2. Payoff after implementation restrictions are applied

1.2 Stakeholders

The newly created marketplace website based on the blockchain has only two stakeholders, which simplifies the process a lot and makes errors rarer. The website developers represented by the small group of students are the central entity in the first proof of concept. These developers build the contract to which the joining parties later agree. They are also responsible for maintaining the website and respond to possible bugs.

Investors are also stakeholders of this application. They can develop over-the-counter contracts (OTC-contracts) within the given boundaries from the smart contract. The investors can be split into separate entities. The party who goes Long and the counterpart who goes Short in the contract. Depending on the type of contract the parties hope or expect the price of the chosen underlying to increase or decrease.

1.3 What solutions our program offers

The proof-of-concept solution provides an innovative way to trade derivatives and has many advantages over traditional trading. The most obvious advantage is the missing of an intermediary. Traditionally, the intermediary brings the two separate positions together. The intermediary normally receives a fee for the brokerage services. This fee is not necessary with the blockchain-based solution. Although the fee does not disappear completely, since a small fee to a miner is still required, the website trading platform definitely provides a less costly solution. An intermediary, such as a bank, can also bear the risk of default. This risk of default may depend on the specifics of the contract that directly affect the investor. Ethereum network carries this risk in the case of a blockchain-based trading platform. This means that the risk of failure is not only borne by one institution but is spread over many small participants.

Traditional trading also allows over-the-counter transactions which do not require an intermediary. Such contracts need to be highly specified to avoid uncertainties for special events. This can be avoided with our platform since the frame is given by the code in the smart contract while still keeping flexibility

in defining the parameters such as maturity, underlying, and strike price. The payment of the counterparty is also enforced by the smart contract due to the depot. This mitigates the risk that the opposite position does not fulfill its obligation. Furthermore, over-the-counter contracts are hard to be regulated by a central entity such as the FINMA. With our solution, the regulation can be written in code and be monitored 24/7 over the blockchain, which should ultimately allow the financial system to be more stable.

The trading platform further allows trading not only on weekdays but also on the weekends. There are no limits due to exchange opening hours. This allows a smoother path of prices since information is fed into the markets continuously.

The current solution not only allows stocks such as Apple and General Motors as underlying but also exchange rates of currencies. For example, one could join a contract on the exchange rate of Bitcoin to USD without owning either of them. The limited down and upside potential further reduce risk since the worst possible scenario can already be quantified.

2 Solution

2.1 Derivative and Share struct

Each of our Derivatives which are created has the following struct as indicated below. The bet is the bet amount in Ether which is to be paid by each participant in order to participate in a bet. The *activationTime* is set at the exact time and date the bet is created. The expiration is the time it takes from that *activationTime* until that bet expires. The strike indicates the strike price in USD. The strings lower and higher are the addresses of both participants depending on their call below or above the strike price. Note that the addresses are both saved in the address payable fields but this time not as string types, as conversions from strings to address types are a hassle. Open indicates whether the derivative is currently open or closed. The *etherAmount* value is the amount of Ether that is put into the Derivative and that number is kept updated depending on how much ether was paid into or out of the contract.

Listing 1.1. Derivative struct

```
struct Derivative {
    uint id;
    uint bet;
    uint activationTime;
    uint expiration;
    uint strike;
    string lower;
    string higher;
    bool open;
    uint underlying;
    uint etherAmount;
```

```
    address payable lowerAddress;
    address payable higherAddress;
}
```

The share struct is the underlying share and defined as indicated below. The name is simply the name of the underlying share, like "Bitcoin" for example. Price is the price of the underlying share which can be either updated or fetched from an API and abbr is the abbreviation like "BTC" for Bitcoin.

Listing 1.2. Share struct

```
struct Share {
    string name;
    uint price;
    string abbr;
}
```

2.2 Smart Contract

In order to start a bet the function *createBet* was created as listed below. The bet can only be opened if the underlying value of the message is equal to the ether amount specified in the bet parameter. The *activationTime* is set accordingly to the exact time and date the bet was opened and according to that the expiration time of the bet is set. A new derivative is then created in the smart contract and in order to keep track of the number of derivatives created the *derivativeCount* is updated.

Listing 1.3. createBet function

```
function createBet(uint bet, uint expiration, uint strike,
    string memory lower, string memory higher,
    uint underlying) public payable {
        uint betInWei = bet * 1000000000000000000;
        require(
            keccak256(abi.encodePacked((msg.value)))
            == keccak256(abi.encodePacked((betInWei))),
            "Please send Ether which is equal to your bet
            amount"
        );
        bool open = true;
        uint activationTime = block.timestamp;
        uint expiration = activationTime + expiration;
        uint etherAmount = msg.value;
        derivatives[derivativeCount] = Derivative(
        derivativeCount, bet, activationTime, expiration,
        strike, lower, higher, open, underlying,
```

```
        etherAmount, payable(msg.sender),
        payable(msg.sender));
        derivativeCount ++;
        }
```

The next main function in the smart contract is the *participateInBet* function which is used, as the name indicates, to join a bet. Multiple checks are done in order to ensure the validity of the participation. A valid open bet has to be selected, the opposite side of the bet has to be selected and the correct amount of ether has to be sent with. Only if all these checks are valid the *etherAmount* in the derivative is upped, the bet is indicated as closed and the opponent address is updated in the derivative.

The last main function for the derivatives themselves is the *checkDerivative* function. This function checks whether a bet has expired or if the bet has been open for more than ten days. If a bet is closed and has expired the price of the underlying asset is checked and the full bet amount is paid out to the winner which has selected the correct bet. If the bet has been open for more than ten days and nobody has joined the bet and closed it the amount is simply returned to the address which has opened the bet. This is made to ensure no bet is open forever and the person which has opened the bet never gets his money back.

Listing 1.4. setPrice function

```
function setPrice(uint _price, string calldata _shareName)
external returns (string memory) {
    bool changedprice = false;
    for (uint i=0; i < shareCount; i++){
        if (keccak256(abi.encodePacked((shares[i].name)))
        == keccak256(abi.encodePacked((_shareName)))){
            shares[i].price = _price;
            changedprice = true;
        }
    }
    if (changedprice){
        return append("Changed price to ",
        uintToString(_price));
    } else {
        return "Error: Could not find the share";
    }
}
```

Further, an important part of our smart contract functions is the way to change the share struct. The function above is the main way we are changing the price of the underlying shares which are then used to select a winner in the bets and are therefore crucial in our application. The price of each share can be changed manually or via an API the price of the underlying shares can be changed and then saved into the smart contract with the help of this function.

2.3 Front-end

The front-end is a web application built with the *React* [2] library. The *Graphical User Interface* (GUI) shows four main components; (i) the *Navbar*, (ii) *Open Bets*, (iii) *Closed Bets*, and (iv) an *Admin Page* as illustrated in figures 3 and 4. In section 2.4, it will be explained how to create new bets using the *Navbar*, joining open bets by the *Open Bets* component, and interpreting the entries in the *Closed Bets* component. In section 2.5, the admin page is explained in more detail in order to set the prices of underlying manually by an administrator.

2.4 Derivative Trading

In order to create a bet, the *Navbar* provides six input fields. The *Higher/Lower* input allows the user to guess if the value of an underlying is increasing or decreasing. For the second input, the user shall provide its *Wallet* address to be uniquely identified. The next input is the *Value Bet in ETH*. Here, the user provides the amount of money, Ethereum in our case, for the bet. The *Strike* field specifies the threshold for the underlying price to which the user wants to be above or below. Furthermore, it is mandatory to provide the duration of the bet in the *Expiration* input field. Here, the user needs to provide the duration time in seconds. In the last field, the user chooses the *Underlying* for the bet. A selection of various underlyings is provided by the smart contract. By clicking on the *Submit* button, the data put inside the form will be pushed to the smart contract on the blockchain.

The *Open Bets* component shows already created bets but without an adversarial party. The *Bet* field shows the amount of Ethers for the bet a user can participate with. In the second field, the *Expiration Date* is shown which specifies how long a bet can be valid. In the third field, the *Underlying* is shown on which the bet relies on. The *Strike* field indicates the price which indicates the winning condition for the user who is betting for higher prices of the according underlying. In the field of *Adversary Bet*, the user sees if the bet creator speculates for a higher or lower price of the underlying. The *Join* input field allows the user to enter its wallet address for participating in the bet. By clicking on the *Join* button next to it, the smart contract will be reached and receives the data of the participating user.

Below, in the *Closed Bets* component, the user sees similar information as in the *Open Bets* component. This component shows the bets, which have two participating parties, and therefore it is not possible to join. Furthermore, the user sees the wallet address of all participants of the according bet and also which one thinks the price of the underlying will rise or fall.

So the start page is built by three main components that show information in a concise and consistent way. The usage is straightforward and intuitive for the user in order to create a bet, participate in a bet, and see the ongoing bets with already two parties joined.

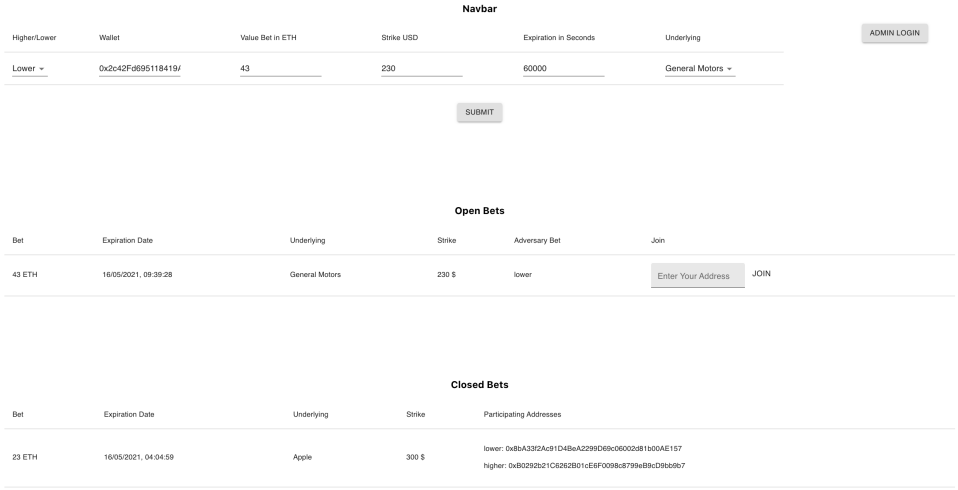

Fig. 3. Start page for creating, joining, and seeing bets.

2.5 Admin Account

The bets rely on the prices of the according underlying. These prices could be fetched from various sources such as from the web. For the scope of this project, the prices are manually set by an administrator. On the start page is an *Admin Login* button (see figure 3), which allows for the admin to popup a special window with a list of underlyings. Now, the administrator is able to put in the new prices for each underlying. The right column also indicates the prices in USD as a reference. Below the input form, the user has the option to click on four different buttons. The *Execute Contracts* button triggers the *checkDerivative* function in the smart contract that verifies if the expiration date is reached and pays out the winner of the bet. With the *Set Current Price* button, the administrator can set the new prices of the underlyings in the smart contract as defined in the form. The user can also go back to the start page by clicking on the *Go To Home* button. Now the administrator can again create, join, or look at the bets that are available in the smart contract. There is also the possibility to log out as an administrator by clicking on the *Logout* button.

2.6 Conclusion

To conclude, the platform presented in this work is a successful proof of concept. The provided solution works within the Ethereum environment with smart contracts and is thus completely decentralized. As such, no central entity or server is needed to trade derivatives. This also allows for trading without any limitations by opening hours of another marketplace. Further, the transparency offered by blockchains and open source code offers a great level of trust in the application.

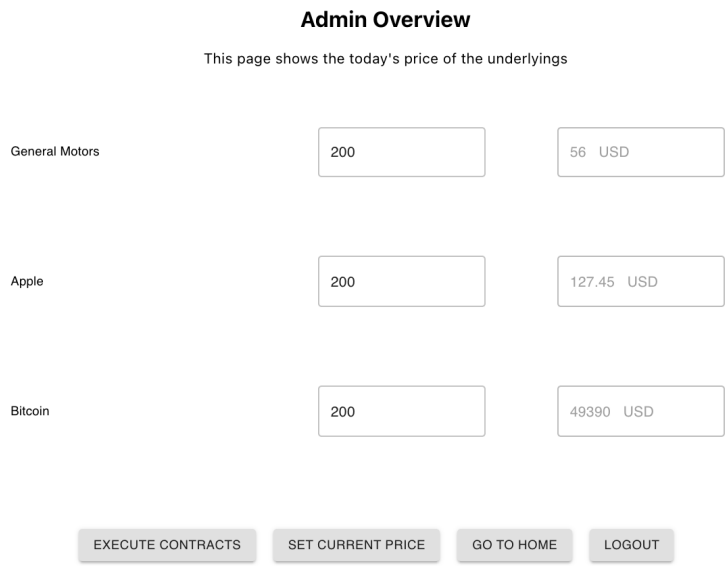

Fig. 4. Admin page for updating the prices of various underlyings.

However, the solution can be extended and improved in future works. For example, a scheduler could be used for the automatic execution of the contracts. Further, financial APIs could be accessed regularly to update the prices of the underlyings. Further, the number of different underlyings can be greatly extended to include more stocks, indexes, commodities, currencies, cryptocurrencies, etc. Also, the application could be extended to allow for more complex derivatives and structured products.

References

1. Postfinance. https://www.postfinance.ch/de/privat/beduerfnisse/anlagewissen/was-sind-derivate.html, accessed: 2021-05-17
2. React. https://reactjs.org/, accessed: 2021-05-15
3. Marc, Chesney, J.K.B.M.S.L.M.: Asset Pricing Finanzderivate und ihre Systemrisiken. Springer Gabler