Permissioned Blockchain for establishing Trusted Ratings

vishal5@stanford.edu

1. Introduction

Ratings form an integral part of validating the worthiness of a financial asset, a movie , a doctor etc. and increasingly dominate our decision making process when choosing between multiple available options. Having a good rating can significantly improve profitability and sales of a product or service however incorrectly tagged rating can result in significant financial loss or inconvenience to the end consumer.
The purpose of the simple blockchain based model described here is to address the threat model wherein a rating agent as an independent entity or colluding with a set of other rating agents influence the correctness of a rating.

2. Why Blockchain?

Blockchain maintains a full history of events / transactions that are securely linked and agreed upon via consensus established by participating nodes. In this specific implementation PBFT consensus is used wherein at max 1/3rd nodes can be byzantine. Rating agents are assumed to be participants of rating firms that are permissioned members of the blockchain. Due to blockchain being permissioned there are no concerns on the identity of a node, However, threat emerging from rating agents trying to tamper the data and or/collude with other rating agents to influence a rating remains. To mitigate this risk , A correctly implemented blockchain would ensure that :-

1. Events / Transactions that are used to compute rating are not tampered with. Block approvals entail 2n/3 votes from n participating rating nodes.

2. Rating provided by a rating agent is based on the same events/ transactions that are available to any other rating agent. (Within an acceptable time frame).

3. Rating Is recaculatable for any historical point in time at any future time. This feature is especially useful if the rating agents are being audited and are asked to prove the legitimacy / accuracy of a historical rating.

Remainder of this paper describes a basic implementation of a blockchain that can be used to establish trusted ratings. Whilst the solution is applicable in any context, the specific scenario used in this paper is Financial Rating.

3. Components

- **Participants (**package **com.project.participant)**
    a. Internal Events Source - this is a data provider of firms internal events , like corporate action, management changes, new projects, etc.

    b. External Event Source - Provides external events like interest rate changes, sector information, global events, etc. For the purposes of this project external events source generates dummy events in response to firm events.

    c. **Rating Agents** - agents that consume internal and external events and provide a point in time rating based on inputs. Each rating agent maintains a local copy of the blockchain and participates in voting for block approvals.

    d. **Byzantine Rating Agent** - is like any other rating agent (extends rating agent class) , however tampers a transaction by changing the event value stored in it.

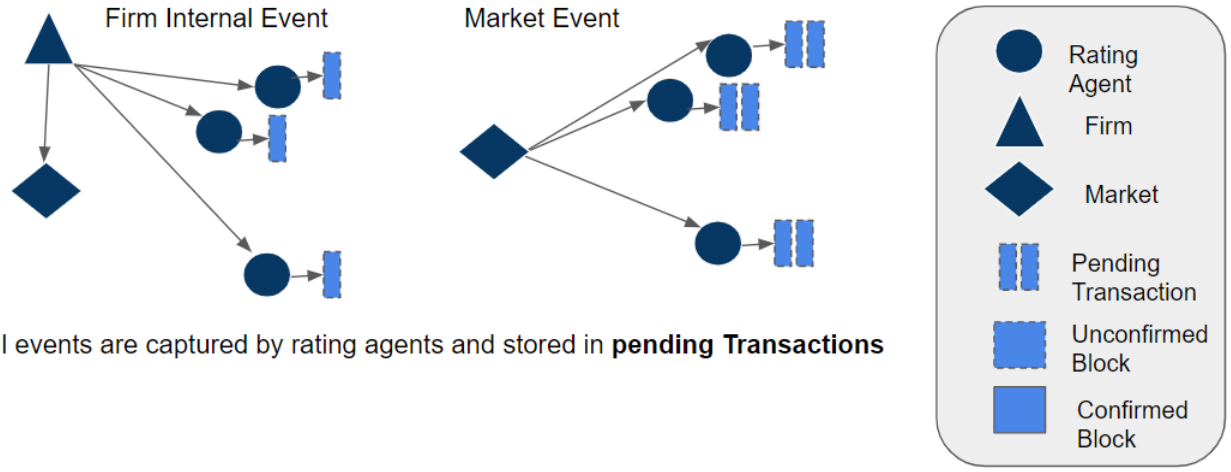
- **Network (**package **com.project.network)**
  Each process is implemented as a multicast node. Networking for the project is built using the java.net package.

- **Consensus (**package **com.project.consensus)**
  Rating Agent coordination and Leader Election is achieved using **Zookeeper**. New Leader is elected at each **Epoch.** A ballot is assigned for each epoch , it holds votes and determines if consensus is achieved.

- **Blockchain (**package **com.project.blockchain)**
  Comprises of Blocks that contain transactions. Blocks (and transactions) contain methods to convert to Json string and back into objects.

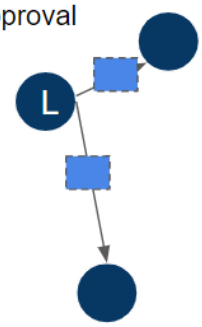
  SHA3-256 scheme is used to compute hashes for blocks and Transactions. Block hash is computed using epoc, MerkleRoot of transactions, previous block hash and timestamp. Blockchain is written out to Rating agents local file system as a json file.

Fig 1 - High level Process Flow

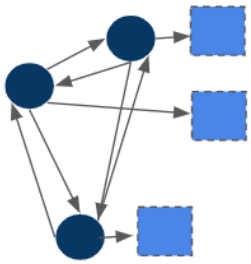


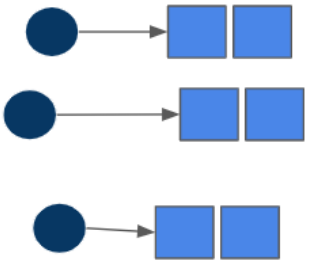
All events are captured by rating agents and stored in **pending Transactions**

New rating agent is elected **leader** at each **epoch**. Elected Leader publishes block for approval



Rating agents **validate** the block and respond with their **vote** of approval if proposed block successfully validates.





If 2n/3 or more approval votes are received, block is **committed** to individual blockchains of rating agents. Transactions included in the block are removed from Pending Transactions Pool.

4. Process

Input Data

1. Internal Events Source generates an event.External Event Source generates event(s) in response to internal events and/or an independent event.

Consume Data

2. Each Rating agent consumes the event(s) and adds to *pendingTransactions* list in their individual localBlockchain (**Note**: Each Rating agent maintains a Local Blockchain)

Leader Election

3. Epoch number is maintained as a **shared counter** managed by zookeeper (**/counter/epoch** Persistent Node). [ref CoordinationService.java]

```
epoch = new SharedCount(client, path: "/counter/epoch", seedValue: 0);
```

4. A new Leader Rating agent is elected at each epoch interval. (leader path in ZK is **/leader/epoch-leade**r). [ref CoordinationService.java]

```
leaderSelector = new LeaderSelector(client,
        leaderPath: "/leader/epoch-leader",
        new LeaderSelectorListener() {
```

Epoch is incremented every 3 seconds, for the duration of this interval the leader assembles transactions to block, proposes it, the block is voted for and committed post consensus. At the end of the epoc leadership is relinquished and the leader goes back into the queue for subsequent election at a later time.

Propose Block

5. Leader looks for max pending transactions by firm id and adds them to a block. New block hash is computed using epoch ID , merkleRoot for included transactions, timestamp, previous block hash [ref: Block.java].

```
String hash = Utils.hash(
        str: previousHash
                + epoch
                + timestamp
                + getMerkleRootHash());
```

The block is published as a proposal to all rating agent nodes. Published block contains hash of current block, previous block hash, timestamp, epoc and a list of transactions.

Vote for Block

6. Rating agents validate the block
    a. Previous block hash should be same as hash of last block on the chain
    b. Epoc number of the current block should be higher than the last block on the chain.
    c. All transactions in the block should be available in *pendingTransactions* of the validating rating agent.

    If all above conditions are met, the Rating Agent publishes their votes to all other ratingAgents. Each rating agent maintains a map of ballotByEpoch, vote count in the ballot for a given epoch is incremented each time the rating agent receives a vote message for the block from other rating agents.

```
public void addVote() { voteCount++; }


/* if votes exceed 2/3 times the total nodes, consensus is achieved.
   vote count maintained at a epoc level by each rating agent
   and is incremented each time a node receives vote on a block in given epoch
 */

public boolean hasConsensus(){
    return voteCount>=Math.ceil(totalNodes*(2/3f));
}
```

Commit Block

7. Leader through its epoch ballot checks for presence of votes >= 2n/3 (n being the number of participating rating agent nodes). If a required amount of votes are received , the leader issues a Prepare request for the block. Individual rating agents (including leader) on receiving this message validate the votes for block in their respective epoch ballots and if all looks good block is committed to their local blockchain. Corresponding transactions are removed from *pendingTransactions* list. Block chain in this implementation is maintained as a json file tagged to individual rating agent, example shown below

{"**epoch**":5,"**previousHash**":"86ae26bd0b10858260a9ddedf7b7bbb54dcd5b6ea71d8ad0f3e3010b715b7d3f","**merkleRootHash**":"934cfeb2330d806310a2ac532022018affe66b362ac02ac20d57db377a4f35d9","**timestamp**":1653500944584,"**Hash**":"f178122ec6ec0b4fdfe8fc0552c8afdc04516521ef31752d205fc5612ece7adc","**transactions":[{**"firmId":"1","eventId":"1-1653500925978","firmEvent":"good","sentBy":"IS"I},{"firmId":"1","eventId":"1-1653500925978","firmEvent":"good","sentBy":"ES","marketEventSourceId":"1","marketEvent":"good"**}]**,"stage":"commit","ratingAgentId":"/rating-agents/0000000000","eventId":"1-1653500925978"}

Note: if the block is not committed in the epoc its proposal was created, it's discarded. Corresponding transactions are picked up in subsequent epochs.

Dealing with Byzantine Agents

8. In the implementation a byzantine node changes the value of a transaction , the block containing a tampered transaction is not accepted by honest rating agents and hence never meets the criterion for consensus. The valid transactions in rejected blocks remain intact in the local pending transaction pool of honest rating agents and are picked up for inclusion in subsequent blocks.

Query

9. Blockchain can be queried via rating agent to:-
    - Fetch blocks by date range.
    - Query all blocks

    Timestamp on block represents the time when it was created; however transactions included in it have a timestamp that represents the time when that specific event was generated in the business context. This bitemporality gives flexibility to query the blockchain in both time dimensions to suit the appropriate use case.

Computing Ratings (**TODO**)

10. Ratings are computed by individual rating agents using information in blockchain and published periodically. Each rating agent can use a different algorithm to arrive at ratings , however rating has to be voted for by other agents who accept a rating block within the a delta threshold of the rating they compute for the same firm/timestamp. Each rating also contains a hash of the codebase that was used to compute it at a point in time; this serves to provide an audit trail with full lineage to how the rating number was arrived at.

5. Conclusion

Permissioned blockchain built on top of a well implemented consensus mechanism is applicable to several use cases that require multiple firms/agents to interoperate. As an example rating agents described in this paper could belong to different companies who work together to ensure a trusted system overall. Incentive models can be created to reward firms correctly approving and/or publishing valid blocks on the network.
Building on concepts applied in this project , A more sophisticated model can be implemented that is applicable to any use case where ratings form an integral part of the consumer decision making process.

6. References

Following papers have been referenced to formalize ideas for the project:

ZooKeeper: Wait-free coordination for Internet-scale systems	https://www.scs.stanford.edu/22sp-cs244b/sched/readings/zookeeper.pdf	
PBFT Practical Byzantine Fault Tolerance	https://www.scs.stanford.edu/22sp-cs244b/sched/readings/pbft.pdf	
Streamlet: Textbook Streamlined Blockchain	https://www.scs.stanford.edu/22sp-cs244b/sched/readings/streamlet.pdf	