University of Zurich^UZH

INDEPENDENT STUDY – Communication Systems Group, Prof. Dr. Burkhard Stiller

Performance Assessment of Cardano

Kürsat Aydinli
Zurich, Switzerland
Student ID: 13-926-910

Supervisor: Sina Rafati
Date of Submission: June 29, 2019

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Independent Study
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Contents

Chapter 1

Introduction

The paper at hand initially originated from the endeavor to compare the performance of different blockchains. In order to do so, various blockchains were screened. With this respect, important aspects were the usability of the blockchain from the users point of view as well as the provision of sufficient tutorials and documentation.
In Particular, following blockchains were assessed in a first phase:

- Elastico
- OmniLedger
- RapidChain
- Zilliqa
- Quarkchain
- RedBelly Blockchain
- Algorand
- Cardano
- Snow WHite
- Ripple
- Openchain
- IOTA
- HydraChain
- MultiChain
- Thundercore
- Elph

- Tendermint
- Harmony-One

Several blockchains were dropped due to missing instructions on how to set up a local network. Furthermore, the focus was primarily set to PoS-based blockchains. In a second run, also BFT-based blockchains should be considered.
Having done the first assessment of the blockchains, following ones were included in the final list:

- PoS-based blockchains
  - Cardano
  - Harmony-One
  - Thundercore
- BFT-based blockchains
  - Zilliqa
  - Tendermint
  - Multichain

The order in the list was set according to the impression of the blockchains usability degree. For instance, Cardano has been chosen as the first blockchain to investigate because of its comprehensive documentation and instructions to easily deploy a private network with API access. Furthermore, Cardano provides quite responsive community support through its Telegram groups, answering requests within hours. When working with a foreign framework, such a characteristic is desirable. Because the examination of the first blockchain took longer than anticipated, the paper only encompasses the assessment of the Cardano network.

Chapter 2

Blockchain Description

Cardano is a Proof-of-Stake (PoS) blockchain initiated and developed by the IOHK foundation [1], a technology company specialized in peer-to-peer applications and cryptocurrencies. Close collaborators in this project are the University of Edinburgh, the University of Athens and the University of Connecticut.
PoS consensus is a natural alternative to Proof-of-Work (PoW) based systems. While in traditional PoW environments, the right to append the next block is determined in a randomized fashion proportionally to the computational power of each miner, the election mechanism in PoS blockchains selects a random miner proportionally to its stake [3].
The currently employed PoS-based consensus mechanism of Cardano is referred to as "Ouroboros Praos", an improvement of their first-generation protocol "Ouroboros".
The protocol of Cardano proceeds in *epochs*, each epoch divided into a fixed number of units called *slots*. During every slot, at most one block might be appended to the blockchain. Thus, there might be slots during which no blocks are generated (see figure 2.1). Note that a slot itself is a relatively short time period, e.g. 20 seconds.

```
+----------+----------+-------+----------+-----------> t
|  slot 0  |  slot 1  |  ...  |  slot N  |

\                                       / \
------------- epoch M ---------------     -- epoch M+1 --
```

Figure 2.1: Sample notion of physical time in Cardano

At every slot, only one stakeholder might be elected as the slot leader. The leader is elected with a probability proportional to its stake registered in the genesis block [7].
Furthermore, it is actively intended by Cardano to have frequently empty slots. This enhances the synchronisation of the blocks in the network. Only a slot leader is aware that it is indeed a leader for a particular slot. The assignment is unknown to all the other participants, until the other stakeholders receive a valid block from the slot leader [1].

[1]https://iohk.io/about/

Although the whitepaper of ”Ouroboros Praos” does not speficy when exactly a block should be generated by the slot leader, the current implementation of Cardano introduces a system paremeter called *network diameter* that approximates the time necessary for the block to be distributed to the whole network. For instance, if this parameter is set to 5 seconds, then the slot leader needs to generate and announce the block the latest 5 seconds before the end of the current slot [5]. Every epoch has its own dedicated genesis block. In contrast to regular blocks, a genesis block does not contain transactions. It contains the epoch index as well as a list of all slot leaders for the upcoming epoch [2].
When a node wants to issue a transaction, it performs following steps: (1) create the transaction and sign it with the private key, (2) send it to all known nodes (neighbors) and (3) save it to the local data.
In turn, each of the nodes neighbors forwards the transaction to its neighbors. Eventually, some slot leader will validate the transaction and include it in the block of the current slot.
Due to the fact that Cardano is based on the Unspent Transaction Output (UTXO) model, each transactions contains a list of inputs and outputs whereas outputs from the current transactions might be used as inputs for the subsequent one [6].
In terms of network topology, Cardano employs three types of nodes: (1) core nodes, (2) edge nodes and (3) relay nodes. The node groups can be characterized as follows [4]:

Core Nodes These are the most important ones. Only these nodes participate in the consensus mechanism and can be elected as slot leaders and will be able to create blocks over this period. To enhance security, core nodes are isolated from the public. This is achieved by putting them inside a perimeter of relay nodes. Only relay nodes are allowed to communicate with core nodes. Furthermore, core nodes are not capable of creating currency transactions, only edge nodes can do so.

Edge Nodes Solely these nodes can issue currency transactions. Since they do not have any stake, they cannot be elected as slot leaders. Besides that, they cannot directly communicate to core nodes, only by means of relay nodes.

Relay Nodes They act as an interface between the core nodes and the public internet. Due to their revealed identity, they might be attacked, but they do not have any stake. Therefore, relay nodes cannot be elected as slot leaders as well.

Chapter 3

Evaliation Scenario

3.1 Cloud Infrastructure

Amazon Web Services (AWS) has been chosen as the main cloud service provider for this project. Initially, the free tier was utilized to run the Cardano blockchain. With the free tier, the user has the opportunity to run VM instances of the type **t2.micro** for 500 hours per month without any charge. This instance type comes with one CPU core as well as 1 GB of RAM.
While experimenting with the setup tutorials for Cardano, it has turned out relatively early that the provided VM capabilities of the free tier are not really sufficient to install and run Cardano. For this reason, the VM of the free tier was shut down and instead of that, an instance of the type **t3.medium** [2 Cores, 4 GB RAM] was utilized.
Furthermore, the VM possesses the AMI of type **Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type** and resides in the availability zone **us-east-1f (North Virginia)**. The reason to use this region is that it resides within the oldest availability zone and as such, software updates get rolled out there first. During the whole evaluation, one VM has been used for testing Cardano.

3.2 Approach

In favour of a successful evaluation, it is crucial to choose an appropriate environment for testing the blockchain. Basically there are two ways to do an assessment of Cardano.
One option is to launch a node in the large network itself (e.g. mainnet or other public testnets of Cardano) including a couple of addresses to accomplish some coin transfers or execute smart contracts. This approach may be suitable to get a overall feeling of working with Cardano. Another incentive for this option might be to assess whether a particular decentralized application (dApp) is compatible with the Cardano ecosystem.
But for the purpose of conducting performance analyses of the entire blockchain, connecting to the mainnet might be an overkill due to the increased complexity implied by the sheer size of the blockchain. Besides that, having the whole mainnet of Cardano under the hood while testing is also likely to increase the storage requirements of the chosen cloud environment.
After consultation with the developer community of Cardano and sharing the background and objectives of the project at hand, it has been decided to create a local (private) Cardano blockchain to do the performance assessment. Not only does this lead to reduced storage requirements, but the complexity of the evaluation is kept to a certain extent due to manageable network size and topology.
According to the developer community, launching a private Cardano network includes, inter alia, following two steps: (1) creation of so-called *genesis data*, (2) launching of the single nodes referring to the previously generated genesis object to make sure they all participate in the same cluster.
Consequently, the next step was to accomplish the two task from above using the official tutorials from the how-to section of the Cardano GitHub repository[1]. Unfortunately, this endeavour was not successful since an end-to-end instruction to accomplish starting a private network was missing and executing the single tutorials did not lead to the desired behaviour. Exploring the Cardano forums revealed that there was one person having a try with the same issue, i.e. creating a local Cardano cluster[2]. That forum post also did not receive a valuable answer. In a final attempt, a Stackoverflow inquiry was set up with the hope of getting some hints with this respect. Sadly, no answers were received.
In summary it can be said that the manual creation of a private Cardano network accross several AWS instances was not successful.
Fortunately, the Cardano repository includes a script to locally launch a standalone demo-cluster consisting of 4 nodes and 12 wallets by default[3]. Throughout the evaluation, this script was used to simulate a local Cardano network. Various properties of the demo-cluster (number of nodes, slot duration, transaction fees etc.) are configurable by an external configuration file.
The downside of using this script is that the generated cluster runs in one single VM without a configuration option to 'control' each single node, i.e. specifying a public IP and port on which the nodes should start. For this reason, the cluster used for this report runs locally on one AWS instance.

[1]https://github.com/input-output-hk/cardano-sl/tree/master/docs/how-to
[2]https://forum.cardano.org/t/how-to-setup-a-private-blockchain-for-cardano/10928/3
[3]https://github.com/input-output-hk/cardano-sl/blob/master/docs/how-to/demo-cluster.md

Once the demo-cluster has been started, common blockchain operations such as sending transactions, retrieving transaction state or blockchain size can be conducted through the Cardano wallet API [4]. The exact API calls will be mentioned at the corresponding locations in the report.

3.3 Challenges

Although it is quite straight-forward to start a private Cardano network, the nature of the demo-cluster poses some obstacles to the evaluation pipeline which is described in more detail in the next section.
The first challenge is related to the spending nature of the blockchain. Cardano is based on the Unspent Transaction Output (UTXO) model, similar to Bitcoin. A single UTXO can be thought of as a bank note. As such, it can only be spent for one transaction at a time. It is not possible to spend a part of an UTXO. As an illustration, one cannot spend a 100 Dollar bill concurrently in two different stores. In this particular case, the person would need to wait to get the change from the first purchase and then spend it in the second store.
Similarly in Cardano, once an UTXO has been spent in the course of a currency transaction, it needs to be wait until the transaction is included in the blockchain and the UTXO balance is re-evaluated.
The current UTXO-statistics for a wallet in Cardano can be queried by following API call:

```
$ curl -X GET \
https://127.0.0.1:8090/api/v1/wallets/Ae2..939q/statistics/utxos \
-H 'Accept:application/json;charset=utf-8' \
--cacert ./state-demo/tls/edge/ca.crt \
--cert ./state-demo/tls/edge/client.pem
```

In the above request, the part **Ae2..939q** denotes the wallet ID for which the UTXO statistics should be retrieved. Upon successful execution, the response looks as follows:

```
{
  "data": {
      "allStakes": 37499998750000,
      "boundType": "Log10",
      "histogram": {
          "10": 0,
          "100": 0,
          "1000": 0,
          "10000": 0,
```

[4]https://cardanodocs.com/technical/wallet/api/v1/?v=1.5.1

```
            "100000": 0,
            "1000000": 0,
            "10000000": 0,
            "100000000": 0,
            "1000000000": 0,
            "10000000000": 0,
            "100000000000": 0,
            "1000000000000": 0,
            "10000000000000": 0,
            "100000000000000": 1,
            "1000000000000000": 0,
            "10000000000000000": 0,
            "45000000000000000": 0
        }
    },
    "status": "success"
}
```

In the JSON response, it can be observed at which 'levels' how many UTXOs exist. Furthermore it can be seen that the wallet is in possession of exactly one UTXO in total: **"100000000000000": 1**
This characteristic holds for all wallets of the demo-cluster and throughout the whole evaluation of Cardano. In practical terms this means that a wallet, after spending some coins, has to wait until the transaction has been approved and included in the blocks. When attempting to issue two immediate transactions without waiting, the second API request returns following error message:

```
{
    "status": "error",
    "diagnostic": {
        "details": {
            "help": "Utxo is not enough fragmented to handle
              the number of outputs of this transaction.
              Query /api/v1/wallets/{walletId}/statistics/
              utxos endpoint for more information",
            "missingUtxos": 1
        }
    },
    "message": "UtxoNotEnoughFragmented"
}
```

A couple of experiments have shown that this waiting time can range from 10 to 30 seconds. In other words, the same wallet can only issue transactions every 10 to 30 seconds.
Logically, this behaviour is insofar unfavourable for stress testing Cardano as a wallet cannot issue transactions continuously (due to the waiting time).

To overcome this obstacle and be able to continuously issue transactions via the API interface, not only one but many wallets have been created in the demo-cluster. This way, transactions can be sent to the network without interruptions. Basically it works by iterating through the wallets and having each wallet committing one transaction, thus using the available UTXO.
The second challenge encountered was related to the logs of the cluster. In order to investigate the exact validation duration of a testrun, it is important to know when the last block was created that still contains transactions. At this point it is noteworthy to mention that, with respect to the evaluation, a transaction is denoted to be validated once it is included in a block.
Although the demo-cluster generates log information about created blocks, these logs were not easily accessible by mainstream editors. The block information is stored in binary format generated through the **RocksDB** database engine.
To decode these kind of logs, a separate application would have to be written which utilises this exact database to retrieve the block information. Because this would be too complex and effortful, another approach has been taken to retrieve the validation duration of a testrun.

The wallet API of Cardano allows the user to retrieve detailed information about the transactions in the blockchain. This can be achieved through following request:

```
curl https://127.0.0.1:8090/api/v1/transactions \
  --cacert ./state-demo/tls/edge/ca.crt \
  --cert ./state-demo/tls/edge/client.pem
```

This request also contains the specific state of a transaction. In Cardano, transactions have following three states: (1) *applying*, (2) *inNewestBlocks* and (3) *persisted*. Here is a sample response of this kind of request:

```
{
    "data": [
        {
            "amount": 100,
            "confirmations": 13,
            "creationTime": "2019-05-14T14:28:41.825214",
            "direction": "outgoing",
            "id": "efojuH...",
            "inputs": [
                {
                    "address": "DdzFF...",
                    "amount": 37499998750000
                }
            ],
            "outputs": [
                {
                    "address": "DdzFF...",
                    "amount": 37499998749900
                },
                {
                    "address": "Aertd...",
                    "amount": 100
                }
            ],
            "status": {
                "data": {},
                "tag": "applying"
            },
            "type": "foreign"
        }
    ],
    "status": "success"
}
```

Given the JSON body, the transaction state can be extracted from the key **"tag": "applying"**. To sum it up, the validation duration has been assessed through constantly querying the transaction state until all transactions have either state *"inNewestBlocks"* or *"persisted"*.

3.4 Evaluation Pipeline

This section describes in detail how the performance of a testrun is evaluated. The workflow consists of 2 main steps which are also a consequence of the challenges mentioned in the previous section. The mentioned scripts can be found in the appendix of the paper. The evaluation pipeline can be thought of as a black box. The input is the number of wallets (needs to be configured before launching the demo-cluster) and the output outlines information regarding the validation duration of all transactions. Hereby, the number of transactions is the same as the number of wallets.
The entire evaluation process is included in the script **do_evalutation.sh**. The script itself consists of following two sub-scripts whereby each script in turn takes care of one of the above mentioned obstacles.

3.4.1 Script 1 - extract_wallets.sh

Starting point for this script is the number of wallets in the deployed Cardano cluster. This property can be set in the configuration file. The main responsibility of the step represented by this script is to generate transactions and send them to the network in a continuous way.
More specifically, following tasks are accomplished during this procedures:

Get All Wallet IDs It is important to retrieve the total of the active wallets which are registered in the running Cardano blockchain. This is accomplished by following command:

```
curl -X GET https://127.0.0.1:8090/api/v1/wallets\
  -H 'Accept:application/json;charset=utf-8' \
  --cacert ./state-demo/tls/edge/ca.crt \
  --cert ./state-demo/tls/edge/client.pem
```

The raw output of the response looks as follows:

```
{
    "data": [
        {
            "balance": 37499998750000,
            "createdAt": "2019-05-14T11:43:07.194824",
            "id": "Aefet....",
            "name": "Imported Wallet"
        }
    ],
     "status": "success"
}
```

The JSON field of interest is **"id": "Aefet...."**. For this purpose, the response of the API call needs to be further processed to retrieve this key. The library **jq** provides mechanisms to filter whole JSON objects. The extracetd IDs of the wallets are stored on an external file, each ID on a separate line.

Issue Transactions Given the file containing all wallets IDs, the next step was to iterate through each wallet and thereby creating a transaction and sending it to the network. Besides that, prior to starting the issuance of the transactions, the current time in seconds is saved in a separate file (needed later for evaluation). The API request for this is the following:

```
curl -s -X POST https://localhost:8090/api/v1/transactions \
  -H "Accept:␣application/json;charset=utf-8" \
  -H "Content-Type:application/json;charset=utf-8" \
  --cacert ./state-demo/tls/edge/ca.crt \
  --cert ./state-demo/tls/edge/client.pem \
  -d '{
  "destinations": [
    {
      "amount": 100,
      "address": "Aefet...."
    }
],
"source": {
   "accountIndex": 2147483648,
   "walletId": "Aefet...."
},
"spendingPassword": ""
}' --http1.1 &> /dev/null
```

In the POST request, the sender as well as the receiver wallet address are the same. Naturally, such a setting (i.e. sending coins to yourself) is not meant to be possible, but in Cardano this was a valid request. However, the result overall is a decrease of the specified amount in the mentioned wallet address.

Having performed these two tasks, the evaluation pipeline continues with the next script.

3.4.2 Script 2 - check_tx_output.sh

Recap that, due to the first script, all transactions have been sent into the network. The main purpose of this script is to assess the total validation time, i.e. how much seconds does it take until all transactions have been included in blocks and thus are part of the blockchain.
Basically this is done through constantly querying the states of all transactions, i.e. *applying*, *inNewestBlocks* or *persisted*. The API request is the same as the one mentioned in the Challenges section.
Every second, the transaction states are retrieved and extracted from the JSON response, again, using the **jq** library. Upon every retrieval, it is checked if there is still an *applying* state represented. If not, this denotes that all transactions are included in blocks and thus, the current timestamp at this point can be written to the external file which also contains the starting time in seconds which was written when issuing the first transaction. This way, it can be assessed how long the validation duration for a particular testrun was. Furthermore, as additional information, the number of blocks created in the demo-cluster at this point is retrieved through:

```
curl -X GET https://localhost:8090/api/v1/node-info \
   -H 'Accept: application/json;charset=utf-8' \
   --cacert ./state-demo/tls/edge/ca.crt \
   --cert ./state-demo/tls/edge/client.pem
```

with following response:

```json
{
    "data": {
        "blockchainHeight": {
            "quantity": 1845,
            "unit": "blocks"
        },
        ...
    },
    "status": "success"
}
```

To sum it up, given a certain number of wallets in the cluster, these two scripts perform the necessary work to observe the total validation duration of all transactions which were issued to the blockchain. Both scripts can be found in the appendix of the report.

Chapter 4

Evaluation Results

This chapter presents the performance results for the evaluation of the Cardano blockchain. The single test scenarios have been executed with 3 different network topologies: 2 nodes, 4 nodes and 7 nodes. Hereby, it is referred to the number of core nodes (see chapter Description). During all tests, only one edge node as well as one relay node were operated. The tables include, inter alia, following information:

- **Sending Duration** [s] - denotes the elapsed time in seconds between the issuance of the first and the last transaction.
- **Validation Duration** [s] - represents the total number of seconds the validation took, i.e. until no transaction is left with the state *applying*.
- **TPS** - validated transactions per second. Calculated by

$$\frac{\#\ Transactions}{Validation\ Duration\ [s]} \tag{4.1}$$

- **Avg. TPS** - represents the average TPS metric of two testruns. Every testrun constellation is evaluated twice and the avgerate TPS values are used in order to draw a comparison between different topologies.

A summary of the tests with a demo-cluster consisting of 2 core nodes is presented in table 4.1.

Table 4.1: Cardano test results for 2 nodes

Testrun	**Slot Duration** [ms]	**# Slots per Epoch**	**# Transactions**	**Sending Duration** [s]	**# Blocks**	**Validation Duration** [s]	**TPS**	**Avg. TPS**
#1	7000	20	12	2	10	27	0,444	**0,436**
#2	7000	20	12	2	10	28	0,428	
#3	7000	20	50	21	19	77	0,649	**0,645**
#4	7000	20	50	21	19	78	0,641	
#5	7000	20	100	80	30	161	0,621	**0,6025**
#6	7000	20	100	80	32	171	0,584	
#7	20000	60	200	337	28	478	0,418	**0,5075**
#8	20000	60	200	335	26	441	0,597	
#9	20000	60	300	888	54	974	0,308	**0,32**
#10	20000	60	300	816	51	903	0,332	

Hereby, 5 different constellations were tested which differ basically in the number of transactions sent to the network. Starting with 12 transactions, being the default setting in the configuration file, the blockchain was also stressed with 50, 100, 200 and 300 transactions (same number as wallets). The reason for not having more than 300 transactions tested is that the demo-cluster runs into some troubles and difficulties while setting up the network. During the launch of the network with more than 300 wallets, the setup exits with a synchronisation error. The Cardano community gave the advise to tweak around with the configuration properties such as slot duration or number of slots per epoch. Even after thorough experimenting with the configuration, it was not possible to get the blockchain running with more than 300 wallets. For this reason, the tests are restricted to a maximum of 300 transactions.
The results from table 4.1 reveal a maximum achieved TPS of 0,645 which was reached when issuing 50 transactions. The lowest performance in terms of the transaction throughput was shown in the default configuration with 300 wallets and 300 transactions. Hereby, the TPS was 0,32. The overall averaged performance is equal to approximately 0,502 TPS. Remarkably is that the issuance duration of the transactions increases drastically as the number of wallets increases. In the case with 12 wallets all transactions were sent within 2 seconds (including the response time of the API request). This results in a TPS sending rate of 6. In contrast, the sending duration of the transactions with 300 wallets was equal to an average of 852 seconds, leading to a TPS sending rate of 0,35 which is significantly lower than the case with less wallets.

This behaviour is also inherent to the evaluations of the other two topologies. With respect to this, it can be noted that the transaction sending time does not correlate linearly with the number of transactions sent.
Table 4.2 gives an overview of the test results for the second scenario of the evaluation, namely the same constellations as in the previous example, but this time having 4 core nodes operating in the Cardano cluster.
The behaviour of the dramatically decreasing sending TPS rate can also observed in this scenario.
With respect to the performance in terms of transaction throughput, the cluster with 4 nodes achieves an overall averaged TPS of 0,483, slightly less than the example with 2 nodes. Hereby, the maximum TPS is equal to 0,588 with 200 transactions whereas the minimum performance can be stated with 0,334 TPS at 300 transactions. The lowest TPS metric in this example is achieved at the same transaction level with the first cluster.

Table 4.2: Cardano test results for 4 nodes

Testrun	**Slot Duration** [ms]	**# Slots per Epoch**	**# Transactions**	**Sending Duration** [s]	**# Blocks**	**Validation Duration** [s]	**TPS**	**Avg. TPS**
#1	7000	20	12	2	10	30	0,4	**0,4305**
#2	7000	20	12	1	10	26	0,461	
#3	7000	20	50	17	26	98	0,51	**0,5205**
#4	7000	20	50	18	24	94	0,531	
#5	7000	20	100	54	34	185	0,54	**0,54**
#6	7000	20	100	54	35	185	0,54	
#7	20000	60	200	206	21	334	0,598	**0,588**
#8	20000	60	200	205	22	346	0,578	
#9	20000	60	300	795	50	896	0,334	**0,334**
#10	20000	60	300	803	50	896	0,334	

The last scenario considers a cluster with 7 nodes and the same number of transactions as the previous two tests (see figure 4.3). Having performing 7 nodes does not lead to surprisingly different performance figures. The TPS values lie in similar value ranges. This time, the maximum achieved TPS was 0,612 with 200 transactions and the lowest throughput was equal to 0,4525 TPS with the default configuration.
A difference in this scenario was that the setup with 300 wallets and just as much transactions failed due to the previously mentioned synchronisation issues.
In general, the experiments with the demo-cluster have shown that, the more complex the network gets, the more difficult and challenging it is to launch it properly and be able to do an evaluation.

Table 4.3: Cardano test results for 7 nodes

Testrun	Slot Duration [ms]	# Slots per Epoch	# Transactions	Sending Duration [s]	# Blocks	Validation Duration [s]	TPS	Avg. TPS
#1	7000	20	12	1	10	27	0,444	**0,4525**
#2	7000	20	12	2	10	26	0,461	
#3	7000	20	50	16	18	91	0,549	**0,5375**
#4	7000	20	50	16	19	95	0,526	
#5	7000	60	100	54	36	186	0,538	**0,545**
#6	7000	60	100	54	35	181	0,552	
#7	20000	60	200	204	23	329	0,62	**0,612**
#8	20000	60	200	208	21	331	0,604	
#9	20000	60	300	sync failed				
#10	20000	60	300	sync failed				

The following section is dedicated to an inter-topology comparison, i.e. for a given number of transactions, the TPS performance is outlined for all network structures.
In the default setting with 12 transactions, it can be seen that the best performance is achieved with the largest network while the cluster with 4 nodes has exhibited the lowest transaction throughput (see figure 4.1).

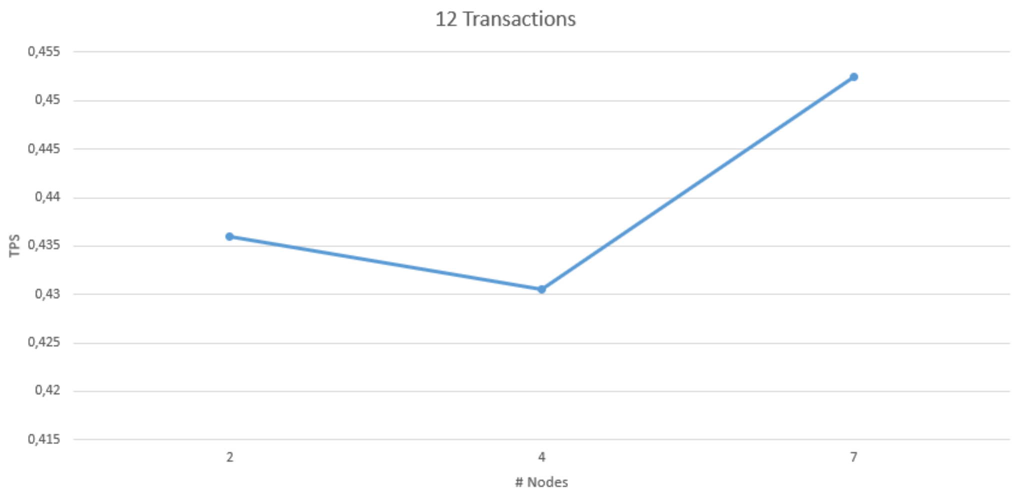

Figure 4.1: TPS assessment with 12 transactions

Figure 4.2 shows the performance comparison for the setting with 50 transactions. It seems that an increase of the network does not lead to a better performance. The more core nodes participate, the lower are the resulting TPS figures.

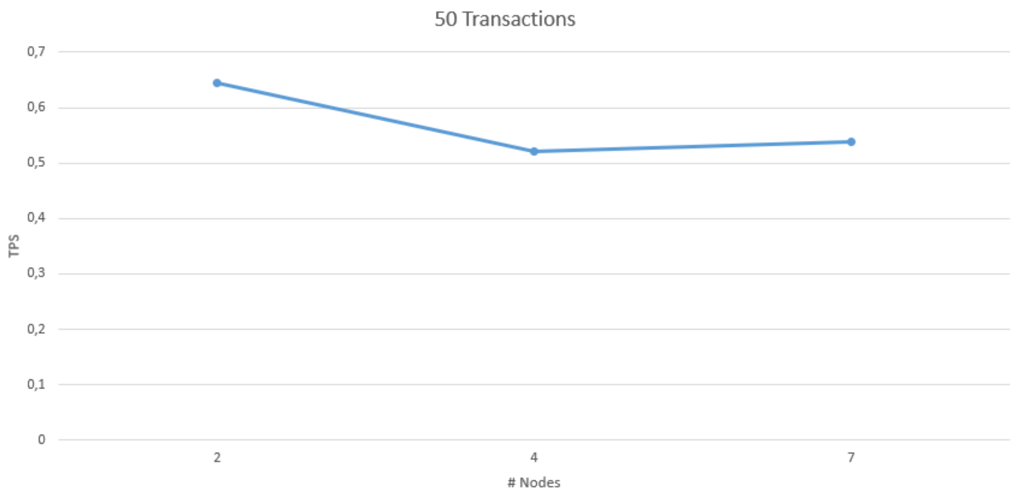

Figure 4.2: TPS assessment with 50 transactions

This effect becomes even more evident in Figure 4.3. The decrease of the transaction throughput is more significant with 100 transactions than with 50 transactions.

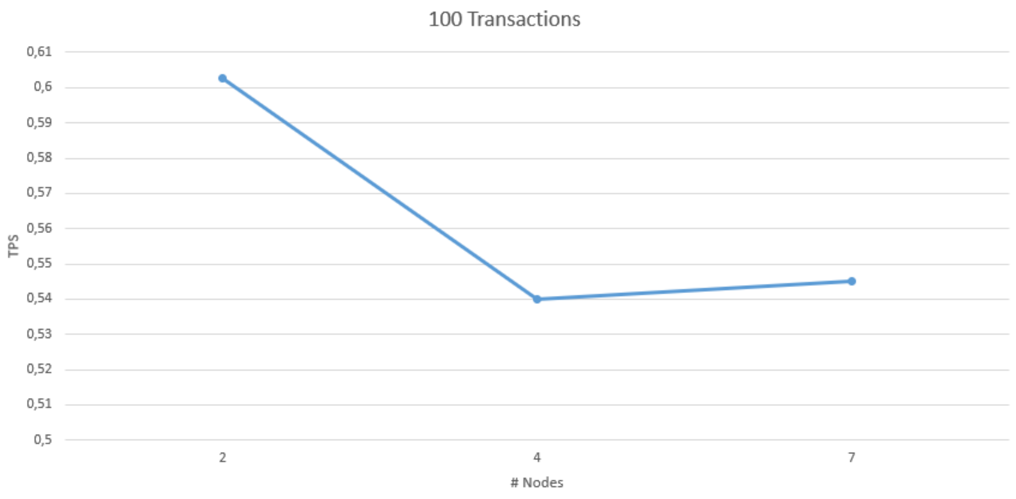

Figure 4.3: TPS assessment with 100 transactions

When issuing 200 transactions to the network, the cluster size has a positive effect on the performance. From figure 4.4 it can be observed that the system scales better than the setup with 50 or 100 transactions.

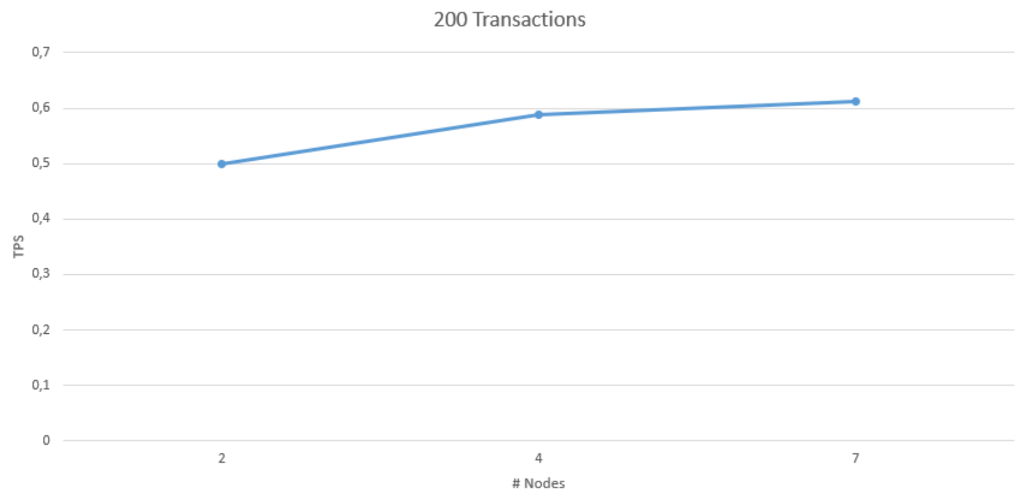

Figure 4.4: TPS assessment with 200 transactions

The experiments have shown that large numbers of transactions (> 50) positively correlate with the scalability of the system in terms of performance gain with increasing cluster size. This conclusion can also be underpinned when looking at the setup with 300 transactions (for which the diagram is omitted due to failure of the run with 7 nodes). The test runs with 300 transactions result in a positive correlation between cluster size and TPS [2 nodes = 0,32 TPS; 4 nodes = 0,334 TPS].
Interestingly, the initial whitepaper of Cardano records quite different performance statistics. A test run with 40 nodes and slot duration of 5 seconds revealed a performance of approximately 257 TPS on average [7].
Unfortunately, it is not mentioned how many of the three node types (relay, edge, core) the test run involved exactly. Furthermore, it is not specified whether the transaction ingestion was carried out by utilizing the REST API.
Compared with the average performance peak of the evaluations at hand, the results from the whitepaper exhibit roughly 500 times higher values than our assessment.

Chapter 5

Future Work

To enhance the performance evaluation of Cardano, starting from the setup of this paper, following two improvements might be considered:

Increase number of edge nodes As outlined in the description part, only edge nodes are able to issue currency transactions to the network. In order to better stress the demo-cluster, many edge nodes could be employed to promote transaction ingestion at a high rate. In fact, this is crucial to get the most out of a particular cluster. For instance, having blocks fitting 400 transactions and a block interval of 20 seconds, a theoretical transaction throughput of 20 TPS might be reached. The precondition for doing so is to issue the transactions from the edge nodes at a rate of > 20 TPS. With the current setup of one edge node this was not feasible. Adding more edge nodes to the cluster seems to be a way of substantially increase the overall performance.
However, this attempt has some drawbacks in terms of effort. First of all, launching edge nodes in the demo-cluster is not as straightforward as creating core nodes. Adjusting core nodes is simply done by manipulating configuration values. An edge node has to be created manually in the launch script.
Besides that, having several edge nodes in a cluster also leads to a much more complex evaluation pipeline. The reason for this is that every edge nodes has its own 'view' of the system. This means that wallets registered through edge node **E1** are not recognized by edge node **E2**. The same holds for transactions. Issued currency transactions of **E1** cannot be seen by **E2**. This behaviour requires that the user needs to evaluate every edge node in a separated fashion and correctly consolidate the output to be able to provide a statement of the overall network performance.

Connect multiple demo-clusters With respect to increasing the cluster size, it might be worth investigating the launch of multiple demo-clusters (e.g. on different VMs) and then trying to connect them together such that the clusters would form one single network. It would be interesting to see how the performance would be affected when an already running cluster would connect to another, second cluster of the same topology.

Chapter 6

Summary and Conclusions

This project was dedicated to perform a performance evaluation of the Cardano blockchain in terms of achieved transaction validation throughput. To do so, following general steps were taken: (1) start a Cardano network with different constellations, (2) issue transactions to the network and (3) measure the validation duration.
Fortunately, the Cardano repository provided a standalone script which made it easy to launch a demo-cluster. In contrast, the last two tasks required a pipeline with many manual steps which are provided by the scripts in the appendix.
Having done the evaluation of Cardano with the network topology presented previously, the performance was compared to the official test run documented in the whitepaper. It could be observed that the performance figures are too far apart. The setup from the whitepaper achieved a much better performance than the evaluation setup of this project. This is probably due to the fact that only a single edge node was launched within the scope of this project. As only those nodes can issue transactions, their quantity within a cluster is likely to have a positive impact on the transaction issue rate which in turn needs to be as high as possible to fill each block to its maximum. This is necessary to exhibit high TPS figures overall.
Furthermore, every wallet installed on a computer from a Cardano user represents an edge node. Thus, the mainnet as well as the testnet contain way more edge nodes than the demo-cluster from this project. Due to this, the community can expect from the main- and the testnet also much better performance figures that the setup of the assessment at hand.

Bibliography

[1] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018. https://cryptorating.eu/whitepapers/Cardano/573.pdf.

[2] IOHK. Blocks in cardano sl. https://cardanodocs.com/technical/blocks/. Accessed: 2019-05-16.

[3] IOHK. Cardano settlement layer documentation. https://cardanodocs.com/introduction/. Accessed: 2019-05-16.

[4] IOHK. Cardano sl network topology. https://cardanodocs.com/cardano/topology/. Accessed: 2019-05-16.

[5] IOHK. Differences between the ouroboros protocol and the implementation. https://cardanodocs.com/cardano/differences/. Accessed: 2019-05-16.

[6] IOHK. Transactions in cardano sl. https://cardanodocs.com/cardano/transactions/. Accessed: 2019-05-16.

[7] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017. https://whitepaperdatabase.com/wp-content/uploads/2018/03/cardano-ada-whitepaper.pdf.

List of Figures

List of Tables

Appendix A

Evaluation Setup

A.1 Installation Guidelines

This chapter describes the steps required to run the evaluation pipeline and reproduce the evaluation results. The guidelines assume a Unix environment:

- **Install Nix:**

```
sh <(curl https://nixos.org/nix/install)
 . /root/.nix-profile/etc/profile.d/nix.sh
```

- **Create Nix config file:**

```
cd /etc/
mkdir nix
cd nix
touch nix.conf
nano nix.conf
```

  Once the file is open, paste following content to the file:

```
    build-users-group =
substituters           = https://hydra.iohk.io https://
   cache.nixos.org
trusted-substituters =
trusted-public-keys   =  hydra.iohk.io:f/Ea+s+dFdN+3Y/G+
   FDgSq+a5NEWhJGzdjvKNGv0/EQ= cache.nixos.org-1:6
   NCHdD59X431o0gWypbMrAURkbJ16ZPMQFGspcDShjY=
```

- **Clone Cardano repository:** Assuming git is installed [1]

```
git clone https://github.com/input-output-hk/cardano-sl.git
```

- **Launch demo-cluster:**

```
cd cardano-sl/
nix-build -A demoCluster -o launch_demo_cluster
./launch_demo_cluster
```

The configuration file can be found under *'/nix/store/c818janh7s1h4x6vkswrjlbbix7dh1ig-cardano-config/configuration.yaml'*.
Once the demo-cluster is running and the wallets are importet (can be observed from the logs in the terminal), the evaluation script can be launched:

```
sh do_evaluation.sh
```

For this purpose, the three scripts from the appendix need to be included into the current working directory, i.e. cardano-sl/. The finalization of the evaluation will create the file **duration_check.sh** (same directory). This file logs the starting timestamp of the transaction ingestion, the time when the sending of the trnsactions are done as well as the final timestamp, after all transactions are included in blocks. Furthermore, the file states the total number of blocks generated up until then.

A.2 Tweaking the Demo-Cluster

Manipulating topology properties such as number of cores or wallets require adjustment of the configuration file as well as the launch script of the demo-cluster.

A.2.1 Number of Nodes

To increase or decrease the number of core nodes, open the launch script and adjust following location accordingly:

```
...
cardano-sl-cluster-prepare-environment "DEMO_" --cores 2 --
   relays 1 --edges 1
cardano-sl-cluster-demo --cores 2 --relays 1 --edges 0 &
...
```

[1]https://www.atlassian.com/git/tutorials/install-git

Hereby, the argument **–cores X** needs to be set on both lines with the same number. Furthermore, the configuration file needs to be adjusted on the following line:

```
...
dev: &dev
  core: &dev_core
    genesis: &dev_core_genesis
      ...
      spec: &dev_core_genesis_spec
        initializer:
          testBalance:
            poors:        100
            richmen:      2 #change here
...
```

The porperty **richmen:** should correspond to the same number of core nodes specified previously in the launch script.
Once these two properties are set, the demo-cluster will launch with the newly specified number of nodes. However, the evaluation of this paper has only worked with up to 7 nodes. It might be possible that the configuration file needs more adaptations in order to launch more nodes successfully.

A.2.2 Number of Wallets

As mentioned above, the number of wallets has determined the number of transactions which were sent to the network in the course of the test runs. This configuration also requires changes in both files. In the launch script, edit following line:

```
...
echo "Importing 11 poor HD keys/wallet..."
for i in {0..99} #change here
do
...
```

To include more then the 12 wallets by default, increase the upper bound of the for loop to 1 minus the desired value (since the indices are 0-based). For instance, if it is intended to have 100 wallets, then set the upper bound to 99 and so on.
The corresponding manipulation of the configuration file needs to take place at this line:

```
...
dev: &dev
  core: &dev_core
    genesis: &dev_core_genesis
      ...
      spec: &dev_core_genesis_spec
        initializer:
          testBalance:
            poors:         100 #change here
            richmen:       2
...
```

But this time with the difference that the key needs to be set exactly to the desired value. For instance, when aiming to have 100 wallets, then set the key **poors:** to 100.
Note that, in case the values of the configuration file and the launch script do not correspond, the blockchain will stop growing after some height, i.e. no further blocks will be created and thus, pending transactions will never be applied and validated.

Appendix B

Required Scripts

B.1 Script 'do_evaluation.sh'

Listing B.1: 'do_evaluation.sh'

```
#!/bin/sh

echo "Starting Evaluation"
echo "Executing script extract_wallets.sh..."
sh extract_wallets.sh

echo "Execuing script check_tx_output.sh..."
sh check_tx_output.sh

echo "Done. Please check the output file duration_check.sh"
```

B.2 Script 'extract_wallets.sh'

Listing B.2: 'extract_wallets.sh'

```
#!/bin/sh

echo "Extracting all Wallets..."

echo "Get total number of wallets"

curl -X GET https://127.0.0.1:8090/api/v1/wallets\
  -H 'Accept: application/json;charset=utf-8' \
  --cacert ./state-demo/tls/edge/ca.crt \
  --cert ./state-demo/tls/edge/client.pem --output
     all_wallets.json

num_wallets=$(jq '.meta.pagination.totalEntries' all_wallets.
   json)
echo "$num_wallets"

echo "Get total number of pages"

#Round up to the next integer
num_pages=$(( (num_wallets + 50 -1) / 50 ))

echo "$num_pages"

echo "Creating final JSON sceleton"
rm -rf all_wallets.json

echo "{" >> all_wallets.json
echo '"all_wallets":[' >> all_wallets.json

#Loop through every available page and append the JSON data
   to the final output file
for i in $(seq 1 $num_pages); do

curl -X GET 'https://127.0.0.1:8090/api/v1/wallets?page='$i'&
   per_page=50' \
  -H 'Accept: application/json;charset=utf-8' \
  --cacert ./state-demo/tls/edge/ca.crt \
  --cert ./state-demo/tls/edge/client.pem --http1.1 >>
     all_wallets.json

echo "," >> all_wallets.json

done
echo "]" >> all_wallets.json
echo "}" >> all_wallets.json

#Extract the ID elements of the JSON output
grep -Po '"id":.*?[^\\]",' all_wallets.json >
   output_wallet_id.txt

#Beautify the output and cut leading and traling characters
sed '{s/^.\{6\}//;s/.\{2\}$//}' output_wallet_id.txt >
   output_wallet_id_cleaned.txt

file="output_wallet_id_cleaned.txt"

#Write Start Time to the final output
rm -rf duration_check.sh
now=`date +%s`
echo "Start Time:" >> duration_check.sh
echo "$now" >> duration_check.sh

while IFS= read line
do

echo "Creating transaction..."

curl -s -X POST https://localhost:8090/api/v1/transactions \
  -H "Accept: application/json; charset=utf-8" \
  -H "Content-Type: application/json; charset=utf-8" \
  --cacert ./state-demo/tls/edge/ca.crt \
  --cert ./state-demo/tls/edge/client.pem \
  -d '{
  "destinations": [
    {
      "amount": 100,
      "address": "'$line'"
    }
  ],
  "source": {
    "accountIndex": 2147483648,
    "walletId": "'$line'"
  },
  "spendingPassword": ""
}' --http1.1 &> /dev/null

done <"$file"

echo "Waiting for all Transactions to be sent..."
```

```
#wait

echo $'\nAll transactions have been sent to the network!!'
tx_sent=`date +%s`
echo "Finished sending transactions:" >> duration_check.sh
echo "$tx_sent" >> duration_check.sh
```

B.3 Script 'check_tx_output.sh'

Listing B.3: 'check_tx_output.sh'

```sh
#!/bin/sh

echo "Get total number of transactions..."

curl https://127.0.0.1:8090/api/v1/transactions \
  --cacert ./state-demo/tls/edge/ca.crt \
  --cert ./state-demo/tls/edge/client.pem --http1.1 --output
     all_transactions.json

num_transactions=$(jq '.meta.pagination.totalEntries'
   all_transactions.json)

echo "$num_transactions"
echo "Get total number of pages"

#Round up to the next integer
num_pages=$(( (num_transactions + 50 -1) / 50 ))
echo "$num_pages"

while sleep 1;
do

rm -rf all_transactions.json

#Boolean to assess if all transactions are in blocks
file_is_ok="true"
#Boolean to stop checking transactions due to some still
   being applied, jump again to outer while loop and wait for
    1 sec
intercept_check="false"

#Loop through every available page and check the transaction
   states, fetching the newest transactions first
for i in $(seq 1 $num_pages); do

# Do an API call to get all TXs
curl -X GET 'https://127.0.0.1:8090/api/v1/transactions?
   sort_by=DES\[created_at\]&page='$i'&per_page=50' \
  --cacert ./state-demo/tls/edge/ca.crt \
  --cert ./state-demo/tls/edge/client.pem --http1.1 --output
     output_tx.json

#Extract the tag elements of the TX json file
```

```
grep -Po '"tag":.*?[^\\]",' output_tx.json >
   output_tx_cleaned.txt

#Given the cleaned up TX states, check if the file is OK
file="output_tx_cleaned.txt"

echo "checking Transactions for page '$i'"

#Checking the lines of the tx output
        while IFS= read line
        do
                if [[ "$line" == *"applying"* ]]; then
                  echo "$line"
                  file_is_ok="false"
                  intercept_check="true"
                  break
                fi
        done <"$file"

        if [[ "$intercept_check" == *"true"* ]]; then
                echo "intercepting transaction check at page
                   '$i'"
                break
        fi

done

#Check if all Transactions are OK. If so, then write final
   commands to the evaluation script
if [[ "$file_is_ok" == *"true"* ]]; then
      echo "Transactions are all applied!"
      now=`date +%s`
      echo "End Time:" >> duration_check.sh
      echo "$now" >> duration_check.sh

      #Get total number of blocks
      curl -X GET https://localhost:8090/api/v1/node-info \
       -H 'Accept: application/json;charset=utf-8' \
       --cacert ./state-demo/tls/edge/ca.crt \
       --cert ./state-demo/tls/edge/client.pem --output
          total_blocks.json

      echo "Total Blocks created:" >> duration_check.sh
      num_blocks=$(jq '.data.blockchainHeight.quantity'
         total_blocks.json)
      echo "$num_blocks" >> duration_check.sh
      exit 1
fi

done
```