NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science
Bachelor's Programme "Data Science and Business Analytics"

Software Project Report on the Topic:

Blockchain network orchestration to improve performance

Submitted by the Student:

group #БПАД222, 2rd year of study Mazurenko Timofey Matveevich

Approved by the Project Supervisor:

Yanovich Yury Alexandrovich
Research Fellow
Faculty of Computer Science, HSE University

Moscow 2024

Contents

Annotation

This project focuses on automating the process of running performance benchmarks for the blockchain network. The goal is to write the program that will receive certain parameters, deploy the blockchain network with these particular parameters, run benchmark for this network and return the results of the benchmark.

Аннотация

Этот проект направлен на автоматизацию процесса запуска бенчмарков производительности для сети блокчейн. Цель - написать программу, которая будет получать определенные параметры, развертывать сеть блокчейн с этими параметрами, запускать бенчмарк для этой сети и возвращать результаты бенчмарка.

Keywords

Blockchain, Orchestration, Distributed Ledger Technology, Automatic Benchmarking, Performance metrics

1 Introduction

1.1 Relevance

The rapid evolution of blockchain technology has introduced a new era of digital transactions known for being highly transparent and secure. The applications of this technology can be seen in different areas of out life such as finance, healthcare, supply chain management and more. As distributed ledger technology grow and evolve, the ability to easily and efficiently test the performance of these networks under certain conditions is extremely important.

As more and more people start to use blockchain technology, the problem of efficient scalability arises. As networks grow in size and complexity, they must maintain high transaction throughput and minimal latency. Automatized bench-marking will provide a way to systematically measure the efficiency and consequently it will help to optimize the network to meet the demand for scaling.

Furthermore, with the huge rise in the technology of machine learning, the relevance of being able to automatically run huge amount of benchmarks for the given network gives an opportunity to collect extremely valuable and reliable data. This data can be later used to train models that will help to identify optimal deploy parameters for particular blockchain network.

1.2 Goal

The goal is to develop the program that will receive deploy parameters of the network, deploy the network with this parameters, run benchmark and collect efficiency metrics.

1.3 Tasks:

- Learn the theoretical foundations of blockchain technology
- Study the features of Hyperledger Fabric - the blockchain I am going to work with
- Develop the program that will automatically run Hyperledger Caliper benchmarks for Hyperledger Fabric and collect data
- Run multiple benchmarks and collect metrics
- Visualize the received data

2 Basic terms and definitions

- **Benchmark** - act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it. [2]
- **Blockchain network** - decentralized, distributed ledger that records transactions across multiple computers in a secure, transparent, and immutable manner, ensuring data integrity and reducing the likelihood of fraud. [10]
- **Distributed Ledger** - database that is consensually shared and synchronized across multiple sites, institutions, or geographies, accessible by multiple people with a decentralized architecture, allowing for a transparent and immutable record of transactions. [8]
- **Chaincode** - Chaincode is a program, written in Go, node.js, or Java that implements a prescribed interface. Chaincode runs in a secured Docker container isolated from the endorsing peer process. Chaincode initializes and manages ledger state through transactions submitted by applications. A chaincode typically handles business logic agreed to by members of the network, so it may be considered as a "smart contract". [7]

3 Description of functional and non-functional requirements

3.1 Functional requirements

1 User should be able to enter parameter and list of values to iterate over for test

2 After user enters the parameters the program updates json file with the parameter and list of values

3 After json if fully packed with the desirable parameters and values to iterate, user can run the launch script

4 Launching script runs benchmarks for all the given parameters and parameter values in json

5 After the benchmark is finished report files with benchmarks data should be created

3.2 Non-functional requirements

3.2.1 Performance constraints

1 The program should not fail in unexpected situations

2 The program should have minimal impact on system performance

3.2.2 Technical specification for running the program

1 UNIX based OS

2 Latest version of Hyperledger Caliper with Caliper CLI

3 Latest version of Hyperledger Fabric

4 Hyperledger Fabric prerequisites

5 Hyperledger Caliper prerequisites

4 Review and comparison of sources and analogues

Note: My project focuses on building the program that automates the process of running benchmarks for Hyperledger Fabric using Hyperledger Caliper bench-marking tool. There is no analogue for this particular use case, however I will still compare it with some other bench-marking tools.

Table 4.1: Comparison table

	My program	Blockbench	MixBytes Tank
Hyperledger Fabric support	Yes	Yes	No
Automatized benchmark running	Yes	No	No
Variety of performance metrics	Yes	No	No

Judging from the table, we can see that my program is extremely relevant due to fact that no other program on the market serves my specific goal - automatization of the bench-marking process

For comparative analysis, i relied on the following resources:

1 Blockbench website [4]

2 Blockbench GitHub [3]

3 MixBytes Tank GitHub [9]

4 Hyperledger Caliper website [6]

5 Hyperledger Caliper GitHub [5]

6 A Brief Overview of Blockchain Testing and Benchmarking Tools [1]

5 Components of developed solution

5.1 Changing parameters in network configuration

First part of the program is changing parameters of the network. Network configuration is described in the core.yaml file available from fabric/sampleconfig/ directory. Here is a python script that accepts 2 command line arguments: key of the parameter to be changed in core.yaml and the new value for the parameter.

editor.py

```
import yaml
import sys

def edit(value, key):
    with open('fabric/sampleconfig/core.yaml', 'r') as file:
        content = yaml.safe_load(file)

    keys = key.split('.')
    d = content
    for k in keys[:-1]:
        d = d.setdefault(k, {})

    d[keys[-1]] = value

    with open('fabric/sampleconfig/core.yaml', 'w') as file:
        yaml.dump(content, file)


if __name__ == "__main__":
    value = int(sys.argv[1])
    key = sys.argv[2]
    edit(value, key)
```

5.2 Preparing input data for benchmark executing script

As described in the functional requirements section, the launching script will require a json file that describes what parameters to test and which values of these parameters should be tested. For proper execution of editing function, also key of the parameter is required. Hence the following parameters.json structure is required

parameters.json

```json
{
    "parameter_name_1": {
        "key": "one.two.three.parameter_name_1",
        "values": [10, 15, 20, 25, 30, 35, 40, 45, 50],
    },
    ....
}
```

5.3 Collecting benchmark results

After the benchmark is done executing, Caliper generates report.html file. However it is a huge html file that contains a lot of unnecessary information. Hence the parser is needed to retrieve core benchmark metrics. All benchmark metrics then should be stored in parameter_report.csv file. Note that each testing parameter has its own dedicated csv file. Here is the python script that does exactly what is written above.

report_parser.py

```python
from bs4 import BeautifulSoup
import csv
import sys


def parse_results(parameter_value, output_path):
    with open('report.html') as report:
        good_soup = BeautifulSoup(report, 'html.parser')

    row = good_soup.find('td', string='readAsset').find_parent('tr')
    data = [td.text for td in row.find_all('td')][1:]

    row = [parameter_value]
    row += data

    with open(output_path, 'a') as table:
        writer = csv.writer(table)
        writer.writerow(row)


if __name__ == "__main__":
    value = int(sys.argv[1])
    path = str(sys.argv[2])
    parse_results(value, path)
```

As report_parser requires path to output csv file, it would be convenient to put this path in json as well. Thus the updated and final structure of parameters.csv is the following:

parameters.json

```json
{
    "parameter_name_1": {
        "key": "one.two.three.parameter_name_1",
        "values": [10, 15, 20, 25, 30, 35, 40, 45, 50],
        "path_to_output": "path/to/output"
    },
    ....
}
```

5.4 Executing benchmarks on given parameters

As all minor helping python scripts are done, the final bash script can be assembled. It consists of several functions that will be described separately.

5.4.1 Running benchmark

Execute_benchmark function can be divided into several steps:

1 Building Hyperledger Fabric (required every time some parameter value is changed)

2 Creating channel and deploying chaincode to the channel

3 Launching benchmark and handling the report

start.sh: execute_benchmark()

```bash
function execute_benchmark() {
    cd ../fabric
    make clean docker-clean peer-docker orderer-docker tools-docker docker-thirdparty docker native

    cd ../fabric-samples/test-network
    ./network.sh up createChannel
    ./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-javascript -ccl javascript

    cd ../../caliper-workspace
    npx caliper launch manager --caliper-workspace ./ --caliper-networkconfig networks/networkConfig.yaml --caliper-benchconfig benchmarks/myAssetBenchmark.yaml --caliper-flow-only-test

    mv report.html ../HLC_automatisation/
    cd ../HLC_automatisation/
}
```

5.4.2 Shutting down the network

Little function that shuts down the network after benchmark is executed:

start.sh: shut_down()

```
function shut_down() {
    cd ../fabric-samples/test-network
    ./network.sh down

    cd ../../HLC_automatisation/
}
```

5.4.3 Adding everything together

The following is the wrapper function that accepts 4 command line input variables:

- Name of the parameter to be changed
- Key of this parameter in core.yaml
- New value to assign for the parameter
- Path to output.csv for specific parameter

Then it uses editor.py for given input to change the parameter value, runs execute_benchmark function, utilizes parser script to collect the data, runs shut_down function to shut down the network.

start.sh: edit_execute_parse()

```
function edit_execute_parse() {

    local parameter=$1
    local key=$2
    local value=$3
    local output_path=$4

    echo "Executing function for parameter: $parameter with value: $value"

    python3 editor.py "$value" "$key"

    execute_benchmark

    python3 report_parser.py "$value" "$output_path"
    rm report.html

    shut_down
}
```

5.4.4 Resetting parameters to default values

Little function that resets core.yaml to default state. It is required, because after all the values for a single paramter are tested, this parameter value should be set to its initial state.

start.sh: reset_core()

```
function reset_core() {
    rm ../fabric/sampleconfig/core.yaml
    cp ../fabric-samples/config/core.yaml ../fabric/sampleconfig/core.yaml
}
```

5.4.5 Executing benchmarks

Finally, parameters.json is unpacked. Little for loop iterates over every given parameter and the benchmark for each value of the parameter is executed.

start.sh: final for loop

```
json_file="parameters.json"
parameter_names=$(jq -r 'keys[]' "$json_file")

for param in $parameter_names; do
    key=$(jq -r --arg param "$param" '.[$param].key' "$json_file")
    values=$(jq -r --arg param "$param" '.[$param].values[]' "$json_file")
    output_path=$(jq -r --arg param "$param" '.[$param].path_to_output' "$json_file")

    for value in $values; do
        edit_execute_parse "$param" "$key" "$value" "$output_path"
    done

    reset_core
done
```

5.5 Adding and deleting parameters

When i was testing my code, i was playing with different parameters. While doing so, i found it pretty inconvenient. Every time the parameter is added to parameters.json, i had to create specific output.csv table and edit json by hand. Same for scenario when parameter is to be removed. Hence i made python script that automatically manages csv tables and edits parameters.json. It consists of several sub-functions:

Creating output file

json_modifier.py: create_output_file

```
def create_output_file(path, param_name):
    with open(path, 'w') as table:
        table.write(f'{param_name},Succ,Fail,Send rate,Max latency,Min
        ↪  Latency,Avg Latency,Throughput\n')
```

Adding new parameter to json

json_modifier.py: add_field

```
def add_field(param_name, key, values):
    with open('parameters.json', 'r') as parameters:
        data = json.load(parameters)

    path_to_output = f'path/to/{param_name}_reports.csv'

    data[param_name] = {
        "key": key,
        "values": values,
        "path_to_output": path_to_output
    }

    with open('parameters.json', 'w') as file:
        json.dump(data, file, indent=4)

    create_output_file(path_to_output, param_name)
```

Deleting parameter from json

json_modifier.py: delete_field

```python
def delete_field(param_name):
    with open('parameters.json', 'r') as parameters:
        data = json.load(parameters)

    if param_name not in data:
        print(f'There is no parameter with name {param_name} in json')
        return

    path = data[param_name]['path_to_output']
    os.remove(path)

    del data[param_name]

    with open('parameters.json', 'w') as file:
        json.dump(data, file, indent=4)
```

Finally, after executed, it allows for quick and easy removal and addition of parameters

json_modifier.py: main

```python
if __name__ == "__main__":
    cmd = sys.argv[1]

    if cmd == 'del':
        delete_field(input('Input name of parameter to delete: '))
    elif cmd == 'add':
        param_name = input('Input name of parameter to add: ')
        key = input('Enter key of parameter: ')
        values = list(map(int, input('Enter list of values to test: ').split()))
        add_field(param_name, key, values)
    else:
        print('Invalid command')
```

6 Sample launch and brief analysis

To test my solution i have tested four parameters. My parameters.json looked like that:

parameters.json

```
{
    "batchSize": {
        "key": "peer.gossip.state.batchSize",
        "values": [10, 15, 20, 25, 30, 35, 40, 45, 50],
        "path_to_output": "benchmark_result/batchSize\_reports.csv"
},
    "maxBlockCountToStore": {
        "key": "peer.gossip.maxBlockCountToStore",
        "values": [10, 15, 20, 25, 30, 35, 40, 45, 50],
        "path_to_output": "maxBlockCountToStore\_reports.csv"
    },
    "maxPropagationBurstLatency": {
        "key": "peer.gossip.maxPropagationBurstLatency",
        "values": [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
        "path_to_output": "maxPropagationBurstLatency\_reports.csv"
    },
    "pullInterval": {
        "key": "peer.gossip.pullInterval",
        "values": [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
        "path_to_output": "pullInterval\_reports.csv"
    }
}
```

Launching the script with this values yielded to 4 csv files as expected. The following is the brief analysis of the results.

First of all, each benchmark tells us number of failed and successfull actions. Neither one of values led to more than 0 fails. Now we can be sure that there is no parameter value that led to extraordinary benchmark result but failed 99% of the time.

Among other metrics, Throughput of the network is the only metric that was somehow changing, hence let us analyze specifically it.

Table 6.1: Affect of parameters on Throughput

Parameter	Average	Median	Min	Max
batchSize	520.87	523.2	487.7	562.2
maxBlockCountToStore	520.8	528.4	489.0	546.7
maxPropagationBurstLatency	499.62	504.7	457.8	530.2
pullInterval	497.39	504.8	458.5	536.5

Following graphs show the dependence between the parameter value and Throughput metric. Background area showing standard error is added to help distinguish between the variability caused by random factors (for example process running in the background of my PC) and the impact of the parameter value itself.

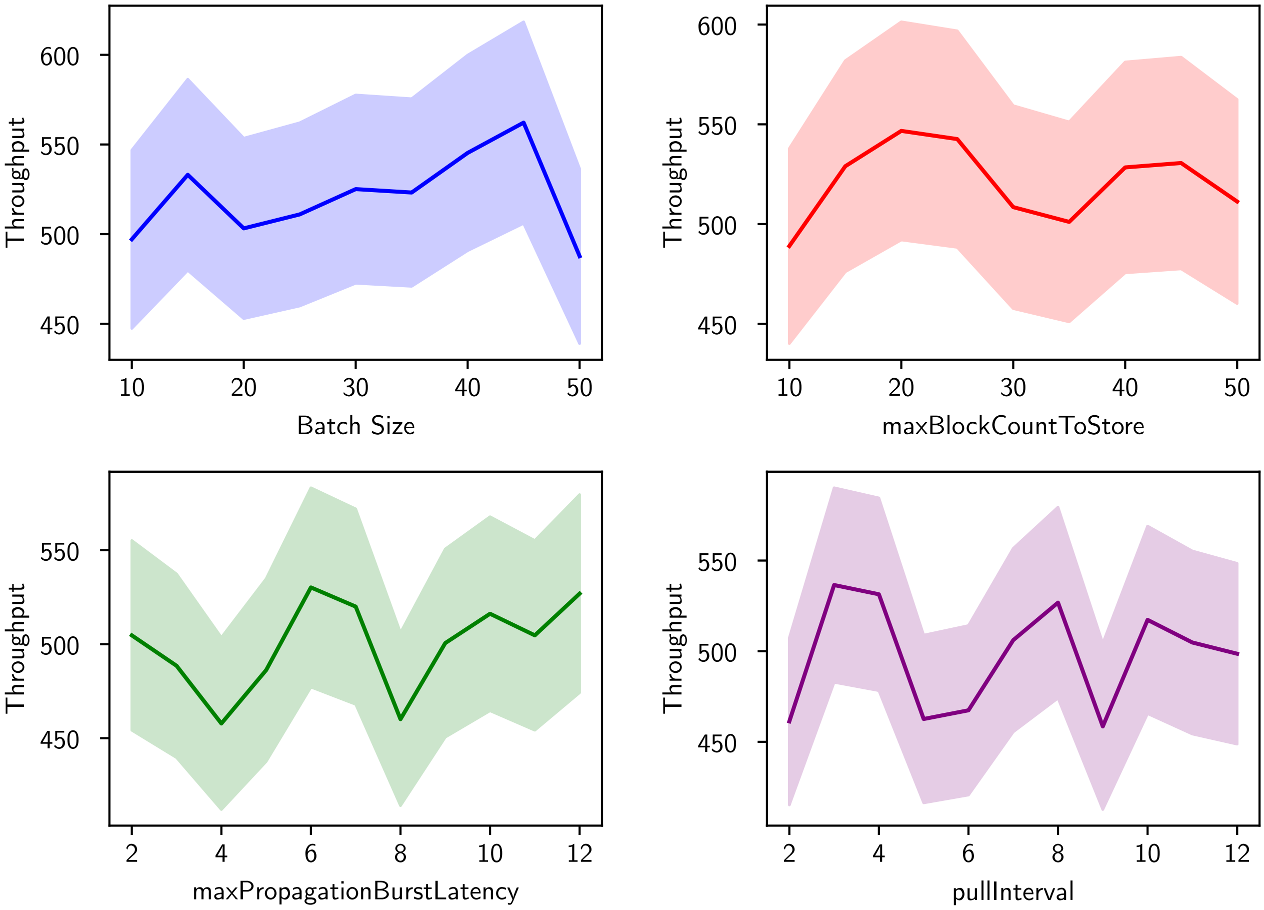

Figure 6.1: Dependence between the parameter value and Throughput

7 Conclusion

All the set goals have been achieved. Previously, collecting benchmarks data for various parameters would require manually changing parameters, building, launching benchmark, collecting data hundreads of times. Now it is as easy, as editing json file via python script and executing a single shell script. Further implications of the built program might be collecting more data. This valuable data can be used to optimize Hyperledger Fabric blockchain network performance via machine learning algorithms or other methods.

References

[1] *A Brief Overview of Blockchain Testing and Benchmarking Tools.* URL: https://hackernoon.com/a-brief-overview-of-blockchain-testing-and-benchmarking-tools-xpk34cc (visited on Feb. 14, 2024).

[2] *Benchmark (computing).* URL: https://en.wikipedia.org/wiki/Benchmark_(computing) (visited on Feb. 14, 2024).

[3] *Blockbench GitHub.* URL: https://github.com/ooibc88/blockbench (visited on Feb. 14, 2024).

[4] *Blockbench website.* URL: https://www.comp.nus.edu.sg/~dbsystem/fabricsharp/#/blockbench (visited on Feb. 14, 2024).

[5] *Hyperledger Caliper GitHub.* URL: https://github.com/hyperledger/caliper (visited on Feb. 14, 2024).

[6] *Hyperledger Caliper website.* URL: https://www.hyperledger.org/projects/caliper (visited on Feb. 14, 2024).

[7] *Hyperledger fabric documentation.* URL: https://hyperledger-fabric.readthedocs.io/en/release-1.3/chaincode.html (visited on May 20, 2024).

[8] Christina Majaski. *Distributed Ledgers: Definition, How They're Used, and Potential?* URL: https://www.investopedia.com/terms/d/distributed-ledgers.asp (visited on Feb. 14, 2024).

[9] *MixBytes GitHub.* URL: https://github.com/mixbytes/tank (visited on Feb. 14, 2024).

[10] *What is blockchain technology?* URL: https://www.ibm.com/topics/blockchain (visited on Feb. 14, 2024).