BlockEmulator User Manual

(Open source version)

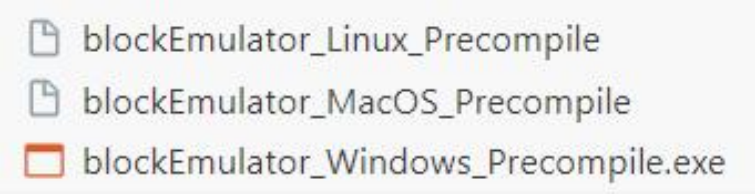
Author: Guang Ye
Mentor: Huawei Huang
HuangLab @ SYSU
http://xintelligence.pro

2024/09/06

2 methods to launch BlockEmulator.

- Difference between these 2 methods:

  - Method 1: Use the pre-compiled executable file
    - No need to install BlockEmulator's dependencies.
    - For easy demonstration and experience.



  - Method 2: Compile and run the source code.
    - Allows you to DIY components of the consensus layer and network layer.
    - Aim for secondary development.

Update in 2024.09.06:
1. Users can now configure parameters in ./paramsConfig.json, including the consensus algorithm (ConsensusMethod), the path to experimental data files (ExpDataRootDir), and dataset files (DatasetFile).
2. Additionally, users can customize the IP addresses of each node in ./ipTable.json.

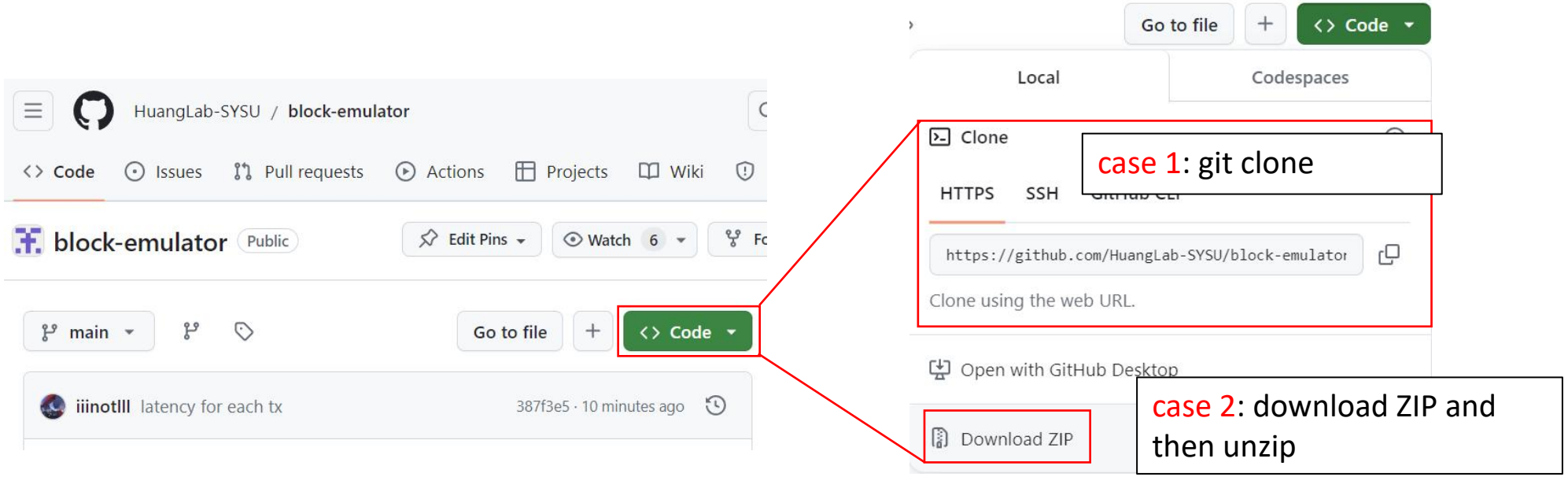
Outline

1. Before running the code: Preparation
2. While running the code: Start BlockEmulator
3. After running the code: Data collection
4. Additional: Install Go environment and modify parameters to run blockEmulator

Preparation - Step ①: Pull the source code

- Pull / download the source code of BlockEmulator from GitHub: https://github.com/HuangLab-SYSU/block-emulator

Click the "code" button in the red box to access the code:



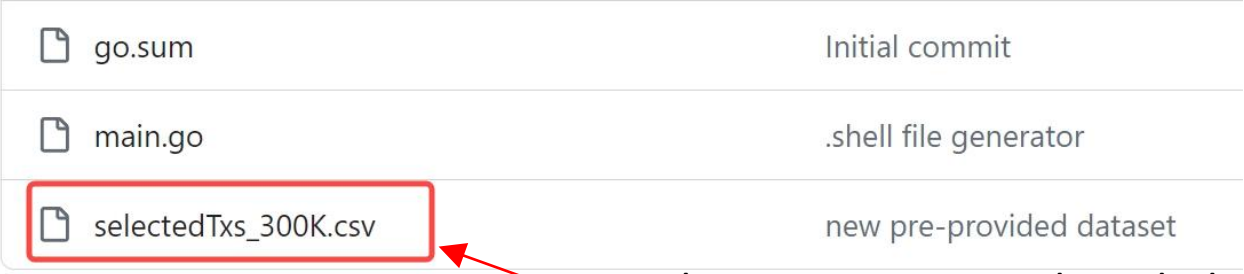
Preparation - Step ②: Download the dataset

The experimental dataset comes from: xblock

(https://xblock.pro/#/dataset/14).

For convenience, the open-source code currently includes a dataset with 300,000 transactions. This dataset was obtained during the "Pull the source code" process on the previous page.

Therefore, for small-scale experiments, users do not require downloading the complete dataset from xblock.



The open-source code includes a dataset with 300,000 transactions.

Outline

1. Before running the code: Preparation
2. While running the code: Start BlockEmulator
3. After running the code: Data collection
4. Additional: Install Go environment and modify parameters to run blockEmulator

Start - Step ①: Understand parameters

- In BlockEmulator, parameters are divided into **two categories**:
  - **First Category**: Parameters related to the specific design details of the consensus protocol, network configuration, network scale, choice of consensus protocol, and output directory for experimental results. These can be modified in ./paramsConfig.json or ./ipTable.json, and changes will take effect immediately.
  - **Second Category**: Parameters related to determining node IDs and generating batch files. These can be specified directly in the command line.

Start - Step ①: Understand parameters

First Category: Rewrite various consensus parameters in ./paramsConfig.json

paramsConfig.json > ...

```
1  {
2    "ConsensusMethod": 0,
3
4    "PbftViewChangeTimeOut":20000,
5
6    "ExpDataRootDir": "expTest",
7
8    "Block_Interval": 5000,
9    "BlockSize": 2000,
10   "InjectSpeed": 2000,
11   "TotalDataSize": 160000,
12   "TxBatchSize": 16000,
13
14   "BrokerNum": 10,
15
16   "DatasetFile": "./selectedTxs_300K.csv",
17   "ReconfigTimeGap": 50
18 }
```

- Consensus algorithms, range: [0,4), representing BrokerChain + CLPA, Relay + CLPA, BrokerChain and Relay, respectively.

- Timeout threshold (in ms) to trigger the PBFT view change.

- Output directory for experimental results.

- Block interval.
- The size of block.
- The speed of transaction injection (txs/s)
- Total # of transactions to be injected.
- The # of transactions contained in each batch when sending transaction batches.

- The # of broker accounts (effective only for ConsensusMethod = 0 or 2).

- File path for the dataset.
- The time gap of reconfiguration. (only for ConsensusMethod = 0 or 1)

Start - Step ①: Understand parameters

First Category: Rewrite the IP addresses of each node in ./ipTable.json:

The 1st key in the JSON file represents the Shard ID.

The 2nd key in the JSON file represents the Node ID.

Represents the node (ShardID=2, NodeID=1), which IP is 127.0.0.1:32427.

Specifically, this represents the IP addresses of the Supervisor node.

```
{
    "0": {
        "0": "127.0.0.1:32217",
        "1": "127.0.0.1:32227",
        "2": "127.0.0.1:32237",
        "3": "127.0.0.1:32247"
    },
    "1": {
        "0": "127.0.0.1:32317",
        "1": "127.0.0.1:32327",
        "2": "127.0.0.1:32337",
        "3": "127.0.0.1:32347"
    },
    "2": {
        "0": "127.0.0.1:32417",
        "1": "127.0.0.1:32427",
        "2": "127.0.0.1:32437",
        "3": "127.0.0.1:32447"
    },
    "3": {
        "0": "127.0.0.1:32517",
        "1": "127.0.0.1:32527",
        "2": "127.0.0.1:32537",
        "3": "127.0.0.1:32547"
    },
    "2147483647": {
        "0": "127.0.0.1:38800"
    }
}
```

Start - Step ①: Understand parameters

Second category: Parameters that can be specified in the command line (You can execute --help in the command line to see the functionality of each command)

For example:.\blockEmulator_Windows_Precompile.exe --help

flags

Explanations of flags

```
f D:\workspace_projects\block-emulator\blockEmulator_Windows_Precompile.exe:
  -g, --gen          isGen is a bool value, which indicates whether to generate a batch file
  -n, --nodeID int   nodeID is an Integer, which indicates the ID of this node. Value range: [0, nodeNum). (default -1)
  -N, --nodeNum int  nodeNum is an Integer, which indicates how many nodes of each shard are deployed.  (default 4)
  -s, --shardID int  shardID is an Integer, which indicates the ID of the shard to which this node belongs. Value range: [0, shardNum).  (default -1)
  -S, --shardNum int shardNum is an Integer, which indicates that how many shards are deployed.  (default 4)
  -f, --shellForExe  isGenerateForExeFile is a bool value, which is effective only if 'isGen' is true; True to generate for an executable, False for 'go run'.
  -c, --supervisor   isSupervisor is a bool value, which indicates whether this node is a supervisor.
pflag: help requested
```

If you are **using a Linux or macOS system** and **lack sufficient permissions**, use **chmod +x {filename}** to grant execute permissions.

Start - Step ①: Understand parameters

Second category: Parameters specified in the cmd

This page explains the meaning of the line below in the --help output:

```
-S, --shardNum int        shardNum is an Integer, which indicates that how many shards are deployed.  (default 4)
```

Short form of this flag. Use "-" to specify it.

Flag Name. Use "--" to specify it.

Description / explanation of this flag.

Default value of this flag.

The following two commands are equivalent:

go run main.go -S 2 -N 4 -s 0 -n 0

go run main.go --shardNum 2 -N 4 -s 0 -n 0

Start - Step ②: Launch & Run

Single-node startup：

BlockEmulator can start a **single node** with the following command:

```
blockEmulator_Windows_Precompile.exe -n 1 -N 4 -s 0 -S 4
```

This command starts a node with the following configuration:

- **Network Scale:** A total of 4 shards, with 4 nodes per shard.
- **Node Details:** The node is in shard 0, with a node index of 1.

In this network configuration, there are 4 shards with 4 nodes each, requiring a total of 16 nodes to be started. **Starting each node individually can be cumbersome**.

Therefore, BlockEmulator provides **a batch startup script** to streamline the process.

Start - Step ②: Launch & Run

Batch generation:

The --help command will provide information about the -g flag:

```
-g, --gen                    isGen is a bool value, which indicates whether to generate a batch file
```

Just add -g to the command line to generate a batch file that can start nodes in batch. (**If you are executing this command with an executable file, you also need to add --shellForExe to generate a batch file specifically for .exe files.)

For examples:

```
go run main.go -g -S 2 -N 4
```

- In the generated batch file, each node is started using "go run" (compile and run).

```
.\blockEmulator_Windows_Precompile.exe -g --shellForExe -S 2 -N 4
```

- In the generated batch file, each node is started using the excutable file.

Start - Step ②: Launch & Run

Result of executing the "generate batch file" command:

After running the command to generate the batch file in ./blockEmulator, for example:

.\blockEmulator_Windows_Precompile.exe -g --shellForExe -S 2 -N 4

A .bat file will be generated in the block-emulator root directory (in Windows)①。

WinExe_bat_shardNum=2_NodeNum=4.bat

① **On Linux or macOS systems, a .sh file will be generated.** If permissions are insufficient, use **chmod +x {filename}** to grant execute permissions.

```
~/block-emulator$ chmod +x blockEmulator_Linux_Precompile
~/block-emulator$ ./blockEmulator_Linux_Precompile -g --shellForExe -S 2 -N 4
```

Start - Step ②: Launch & Run

- **Content of the generated batch file:**

Check the contents of WinExe_bat_shardNum=2_NodeNum=4.bat

WinExe_bat_shardNum=2_NodeNum=4.bat

```
start cmd /k blockEmulator_Windows_Precompile.exe -n 1 -N 4 -s 0 -S 2

start cmd /k blockEmulator_Windows_Precompile.exe -n 1 -N 4 -s 1 -S 2

start cmd /k blockEmulator_Windows_Precompile.exe -n 2 -N 4 -s 0 -S 2

start cmd /k blockEmulator_Windows_Precompile.exe -n 2 -N 4 -s 1 -S 2

start cmd /k blockEmulator_Windows_Precompile.exe -n 3 -N 4 -s 0 -S 2

start cmd /k blockEmulator_Windows_Precompile.exe -n 3 -N 4 -s 1 -S 2

start cmd /k blockEmulator_Windows_Precompile.exe -n 0 -N 4 -s 0 -S 2

start cmd /k blockEmulator_Windows_Precompile.exe -n 0 -N 4 -s 1 -S 2

start cmd /k blockEmulator_Windows_Precompile.exe -c -N 4 -S 2
```

Sequence matters. In code, using sleep for seconds.

Start the non-leader nodes in **shard 0**.

Start the non-leader nodes in **shard 1**.

Start the leader nodes in each of the two shards.

Start the supervisor node, which will inject transactions.

Start - Step ②: Launch & Run

- **Start .bat/.sh file：**

  The .bat file can be run by double-clicking it or using the command line in Windows.

  The .sh file can be run from the command line on Linux or macOS.

Example: To start a .sh file from the command line on Linux or macOS:
Navigate to the blockEmulator directory and enter:

sh Linux_shell_shardNum=2_NodeNum=4.sh

If you receive a file permissions error, use **chmod**:

chmod +x Linux_shell_shardNum=2_NodeNum=4.sh

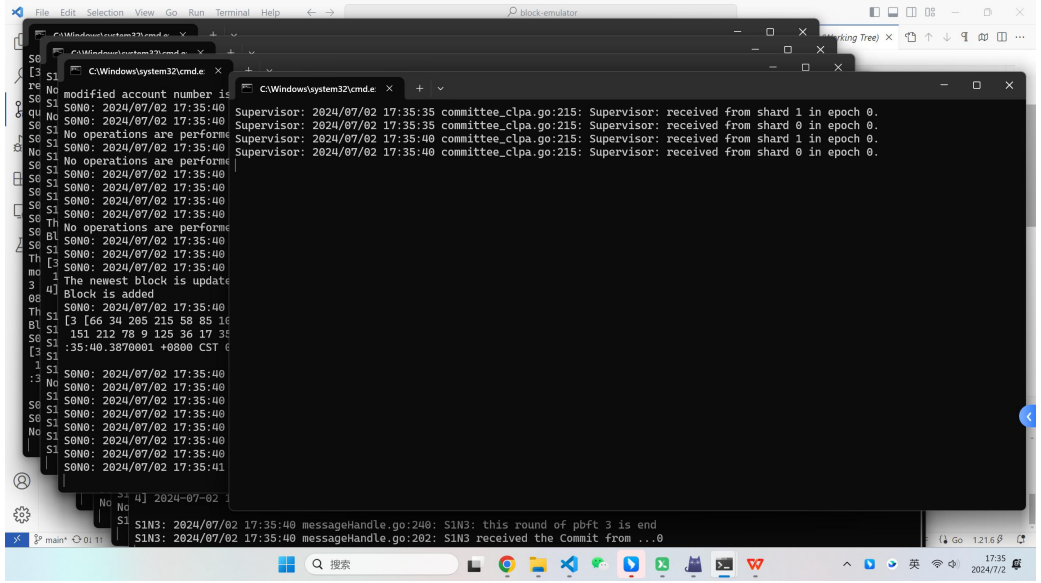
Start - Step ②: Launch & Run

- **The running of BlockEmulator**

  During the execution of blockEmulator, users do not need to take any action; just wait for the results.

  (As shown in the image below, if you are using Windows and start the batch nodes by double-clicking the .bat file, several terminal windows will pop up.

  Each terminal corresponds to one node.)



Start - Step ③: Output logs

S1N2: 2024/07/18 10:32:44 messageHandle.go:240: S1N2: this round of pbft 4 is end
S1N2: 2024/07/18 10:32:44 messageHandle.go:202: S1N2 received the Commit from ...0
modified account number is 2832

(block #) 4 the (MPT's) root (hash) = [8 244 164 12 71 169 53 185 211 146 198 79 206 66 170 254 163 218 69 58 160 191 165 172 130 44 75 52 173 38 30 164]
The newest block is updated

Block is added

S0N1: 2024/07/18 10:32:44 pbftInside_moduleCLPA.go:89: S0N1 : added the block 4...
[4 [93 47 65 159 81 232 192 35 187 28 138 1 39 76 40 20 112 18 57 138 164 46 73 216 6 117 249 144 186 31 243 178] [8 244 164 12 71 169 53 185 211 146 198 79 206 66 170 254 163 218 69 58 160 191 165 172 130 44 75 52 173 38 30 164] 2024-07-18 10:32:43.887244 +0800 CST 0xc00016e240]

S0N1: 2024/07/18 10:32:44 messageHandle.go:240: S0N1: **this round of pbft 4 is end**

The newest block is updated

Block is added

S0N2: 2024/07/18 10:32:44 pbftInside_moduleCLPA.go:89: S0N2 : added the block 4...
[4 [93 47 65 159 81 232 192 35 187 28 138 1 39 76 40 20 112 18 57 138 164 46 73 216 6 117 249 144 186 31 243 178] [8 244 164 12 71 169 53 185 211 146 198 79 206 66 170 254 163 218 69 58 160 191 165 172 130 44 75 52 173 38 30 164] 2024-07-18 10:32:43.887244 +0800 CST 0xc0000f60c0]

Start - Step ③: Output logs

Ending Here

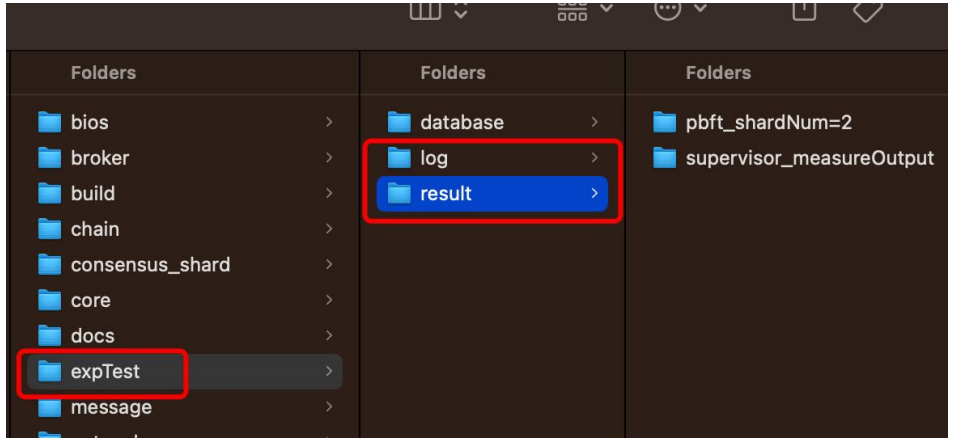
Supervisor: 2024/07/18 10:37:44 supervisor.go:223: Closing...
Supervisor: 2024/07/18 10:37:44 supervisor.go:225: Average_TPS
S0N0: 2024/07/18 10:37:44 messageHandle.go:61: S0N0 get stopSignal in Propose Routine, now stop...
Supervisor: 2024/07/18 10:37:44 supervisor.go:226: [587.7339655544023 699.2846784883424 732.1359262397688 742.7606320250944 359.791113983571 1057.7039834857578] 524.3899226336739

Supervisor: 2024/07/18 10:37:44 supervisor.go:225: Transaction_Confirm_Latency
Supervisor: 2024/07/18 10:37:44 supervisor.go:226: [19.857939453921702 60.99067846275937 105.71421328357414 145.61607771338512 346.77686551225645 457.7541462054386] 115.2762855470001

Supervisor: 2024/07/18 10:37:44 supervisor.go:225: CrossTransaction_ratio
Supervisor: 2024/07/18 10:37:44 supervisor.go:226: [0.39579679374749993 0.2513153030173762 0.1974907126430525 0.17585750744932455 0.6968822010605513 1 160000 48253] 0.30158125

Supervisor: 2024/07/18 10:37:44 supervisor.go:225: Tx_number
Supervisor: 2024/07/18 10:37:44 supervisor.go:226: [32499 35163 36743.5 37419.5 18009.5 165.5] 160000

Supervisor: 2024/07/18 10:37:44 supervisor.go:225: Tx_Details
Supervisor: 2024/07/18 10:37:45 supervisor.go:226: [] 0



Start - Step ③: Running errors

- If **blockEmulator fails to run**, it might be because the ports are occupied by a previous instance of blockEmulator.
    - **Linux/macOS**: Use killall -9 {blockEmulator_MacOS_Precompile} to terminate the process.
    - **Windows**: Close all open terminals.

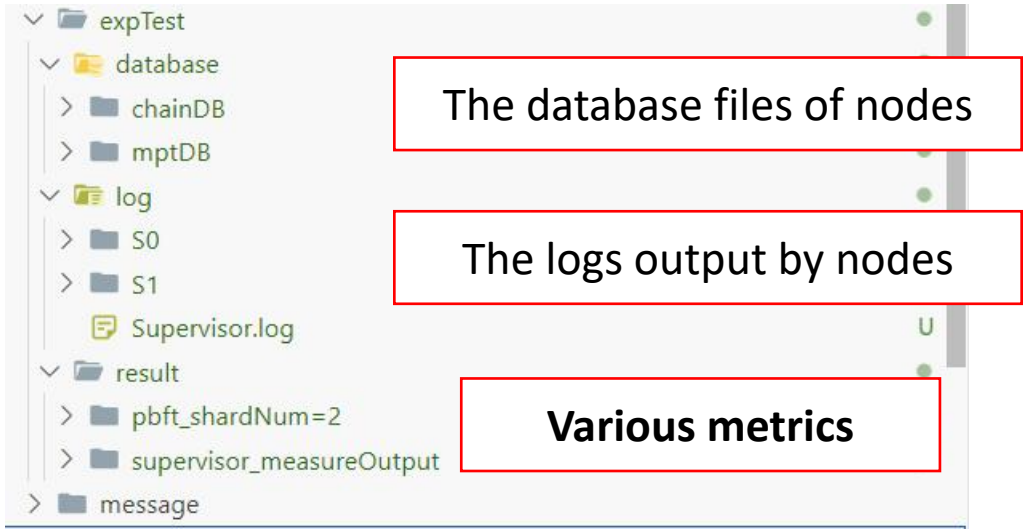
To prevent port conflicts, users can manually modify the IP addresses of each node in ./ipTable.json (See Page 9).

Outline

1. Before running the code: Preparation
2. While running the code: Start BlockEmulator
3. After running the code: Data collection
4. Additional: Install Go environment and modify parameters to run blockEmulator

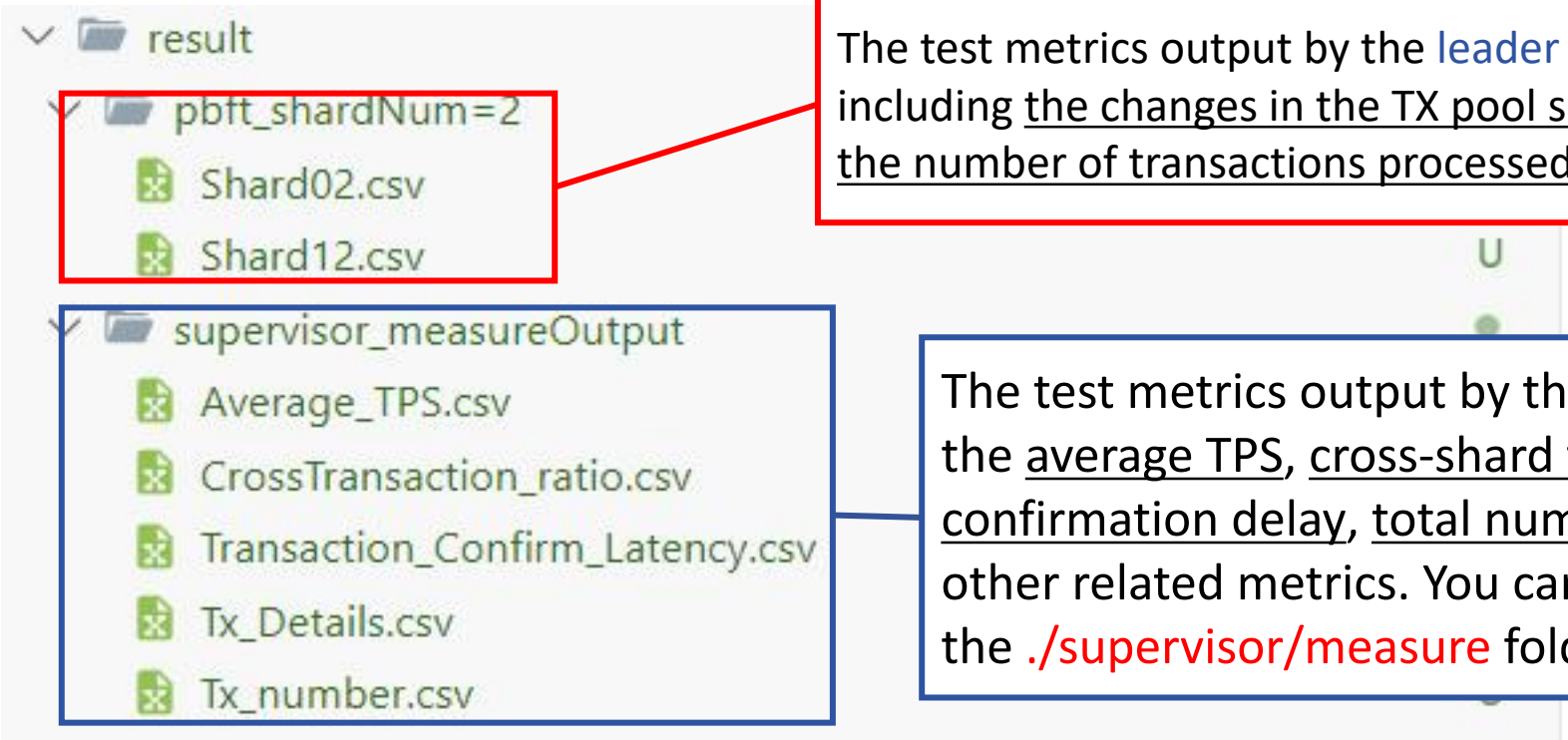
Data collection - Step ①: View the updated dir

- After blockEmulator completes its execution, you will see three folders that have been updated: ./log, ./record, and ./result under the ./expTest directory.
- The ./expTest directory can be specified using the -d option in the command line, allowing each experiment's results to be stored in different directories.



Data collection - Step ②: View ./result

- Among the three folders, ./result records the metrics output by the nodes, which is usually the part we care about the most.



The test metrics output by the leader node of each shard, including the changes in the TX pool size within each shard and the number of transactions processed by each block.

The test metrics output by the supervisor node, including the average TPS, cross-shard transaction ratio, transaction confirmation delay, total number of transactions, and other related metrics. You can refer to the existing code in the ./supervisor/measure folder for customization.

Data collection - Step ②: View ./result

- View the experimental results output by **the leader of each shard**:

  For example, in .\expTest\result\pbft_shardNum=2\Shard02.csv:

	A	B	C	D	E	F	G	H	I	J	K
1	Block Heig	EpochID of	TxPool Siz	# of all T	# of Relay	# of Relay	TimeStamp	TimeStamp	SUM of cor	SUM of cor	SUM of con
2	1	0	0	0	0	0	1.721E+12	1.721E+12	0	0	0
3	2	0	3832	2000	920	0	1.721E+12	1.721E+12	9481584	4361455	0
4	3	0	5630	2000	1001	0	1.721E+12	1.721E+12	19512246	9765860	0
5	4	0	9137	2000	935	168	1.721E+12	1.721E+12	29507817	13792115	2483711
6	5	0	12024	2000	521	884	1.721E+12	1.721E+12	39512859	10286417	17478824
7	6	0	16107	2000	870	370	1.721E+12	1.721E+12	48460113	20697993	9169585
8	7	0	18219	2000	514	598	1.721E+12	1.721E+12	48144923	11119297	17815349
9	8	0	23956	2000	1001	0	1.721E+12	1.721E+12	53237282	26645053	0
10	9	0	28508	2000	777	523	1.721E+12	1.721E+12	65615569	23426944	20803748
11	10	0	31894	2000	988	0	1.721E+12	1.721E+12	56971509	28143714	0
12	11	0	34809	2000	1065	0	1.721E+12	1.721E+12	66968487	35660746	0
13	12	0	38677	2000	554	915	1.721E+12	1.721E+12	91909595	21319645	50155184
14	14	1	41443	2000	641	0	1.721E+12	1.721E+12	97013578	31092840	0
15	15	1	45031	2000	616	23	1.721E+12	1.721E+12	102831814	31528387	1606044
16	16	1	47134	2000	391	925	1.721E+12	1.721E+12	117439906	19705074	63263390

Metric Name

Metric Value

At block height 1, there are no transactions within the block because *the Supervisor has not started injecting transactions at this point*.

Data collection - Step ②: View ./result

- View the experimental results output by **the Supervisor**:

  Example 1, in .\expTest\result\supervisor_measureOutput\Average_TPS.csv:

	A	B	C	D	E	F	G	H
1	EpochID	Total tx ‡	Normal tx	Relay1 tx	Relay2 tx	Epoch star	Epoch end	Avg. TPS of
2	0	31517.5	19035	18651	6314	1.721E+12	1.721E+12	627.18041
3	1	35401.5	26803	12486	4711	1.721E+12	1.721E+12	705.18535
4	2	36065.5	28131	8671	7198	1.721E+12	1.721E+12	719.17138
5	3	39008	34016	7226	2758	1.721E+12	1.721E+12	777.88976
6	4	18007.5	3875	1106	27159	1.721E+12	1.721E+12	359.41138

Metric Name

Metric Value

Data collection - Step ②: View ./result

- View the experimental results output by **the Supervisor**:

  Example 2, in .\expTest\result\supervisor_measureOutput\Tx_Details.csv:

	A	B	C	D	E	F	G	H	I
1	TxHash (By	Tx propose	Block prop	Tx finally	Relay1 Tx	Relay2 Tx	Broker1 Tx	Broker2 Tx	Confirmed
2	5.625E+76	1.721E+12	1.721E+12	1.721E+12	1.721E+12	1.721E+12			153683
3	4.155E+76	1.721E+12	1.721E+12	1.721E+12					75534
4	3.719E+76	1.721E+12	1.721E+12	1.721E+12					72291
5	3.891E+76	1.721E+12	1.721E+12	1.721E+12					106012
6	8.172E+76	1.721E+12	1.721E+12	1.721E+12					137369
7	7.983E+76	1.721E+12	1.721E+12	1.721E+12					134449
8	6.938E+76	1.721E+12	1.721E+12	1.721E+12					143172
9	2.275E+75	1.721E+12	1.721E+12	1.721E+12					4741
10	1.078E+77	1.721E+12	1.721E+12	1.721E+12	1.721E+12	1.721E+12			166928
11	7.994E+76	1.721E+12	1.721E+12	1.721E+12	1.721E+12	1.721E+12			217595
12	8.212E+76	1.721E+12	1.721E+12	1.721E+12					104273
13	6.197E+76	1.721E+12	1.721E+12	1.721E+12					131295
14	8.585E+76	1.721E+12	1.721E+12	1.721E+12	1.721E+12	1.721E+12			214565
15	8.116E+76	1.721E+12	1.721E+12	1.721E+12	1.721E+12	1.721E+12			247035
16	9.381E+76	1.721E+12	1.721E+12	1.721E+12	1.721E+12	1.721E+12			148762
17	1.116E+77	1.721E+12	1.721E+12	1.721E+12	1.721E+12	1.721E+12			243920
18	2.206E+76	1.721E+12	1.721E+12	1.721E+12					48644
19	6.818E+76	1.721E+12	1.721E+12	1.721E+12					55391

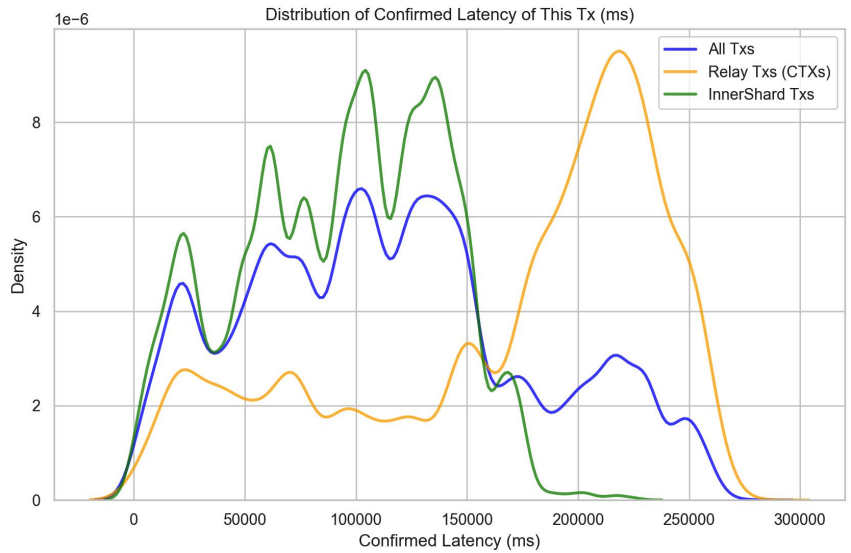
Metric Name

Metric Value

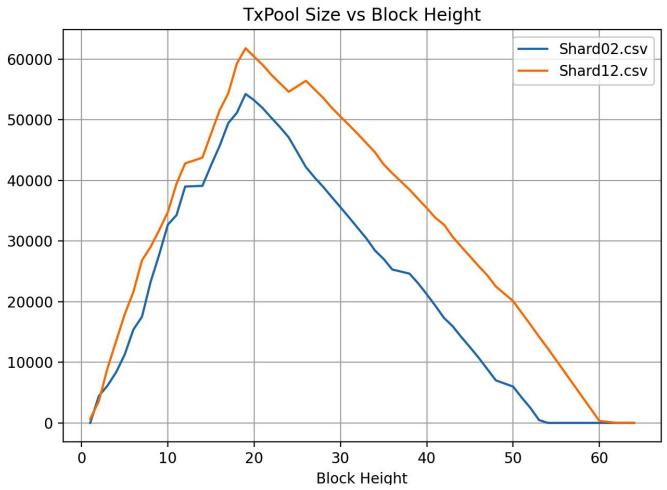
If ConsensusMethod=1 or 3, both of these items related to Broker transactions will be empty.

Data collection - Step ③: Figure Plotting

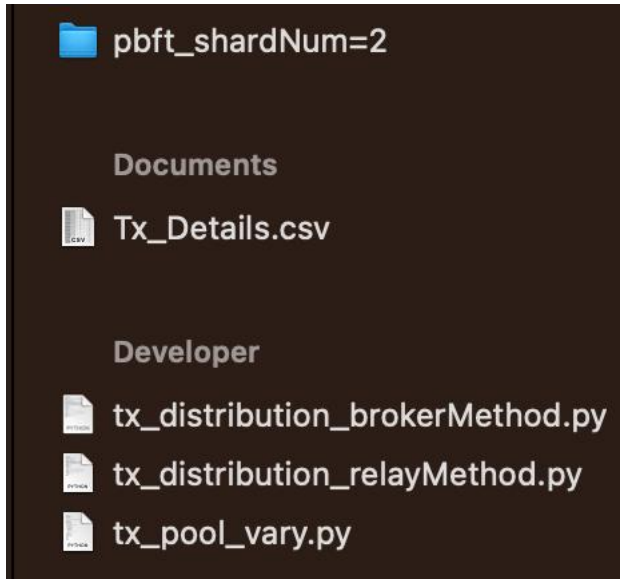
Use the data in ./result to create plots (the plotting code is located in ./figurePlot)



python tx_distribution_relayMethod.py



python tx_pool_vary.py



Outline

1. Before running the code: Preparation
2. While running the code: Start BlockEmulator
3. After running the code: Data collection
4. Additional: Install Go environment and modify parameters to run blockEmulator

Go Compile & Run - Step ①： Configure Go

- Configure Go to Compile & Run blockEmulator

  The official download URL for Go is: https://go.dev/dl/.

  Visit the website and follow the instructions to download and install it.

All releases

After downloading a binary release suitable for your system, please follow the installation instructions.

If you are building from source, follow the source installation instructions.

See the release history for more information about Go releases.

As of Go 1.13, the go command by default downloads and authenticates modules using the Go module mirror and Go checksum database run by Google. See https://proxy.golang.org/privacy for privacy information about these services and the go command documentation for configuration details including how to di the use of these servers or use different ones.

Featured downloads

Microsoft Windows	Apple macOS (ARM64)	Apple macOS (x86-64)	Linux	Source
Windows 10 or later, Intel 64-bit processor	macOS 11 or later, Apple 64-bit processor	macOS 10.15 or later, Intel 64-bit processor	Linux 2.6.32 or later, Intel 64-bit processor	
go1.22.4.windows-amd64.msi	go1.22.4.darwin-arm64.pkg	go1.22.4.darwin-amd64.pkg	go1.22.4.linux-amd64.tar.gz	go1.22.4.src.tar.gz

Go Compile & Run - Step ①： Configure Go

In the blockEmulator directory, run **go get .** or **go mod tidy** to download the modules required to run the code (the specific modules needed can be found in the ./go.mod file).

go.mod

Reset go.mod diagnostics | Run go mod tidy | Create vendor directory

```go
module blockEmulator

go 1.19

Check for upgrades | Upgrade transitive dependencies | Upgrade direct dependencies
require (
    github.com/boltdb/bolt v1.3.1
    github.com/ethereum/go-ethereum v1.11.6
    github.com/spf13/pflag v1.0.5
)
```

The ./go.mod file specifies the required modules.

```
PS D:\workspace_projects\block-emulator> go mod tidy
```

Run **go mod tidy** to automatically download and install the required modules.

Go Compile & Run - Step ②： Modify codes

Now, we can modify the code written in the .go files in BlockEmulator.

consensus_shard > pbft_all > pbftMod_interface.go > ...

```go
package pbft_all

import "blockEmulator/message"

// Define operations in a PBFT.
// This may be varied by different consensus protocols.

type ExtraOpInConsensus interface {
    // mining / message generation
    HandleinPropose() (bool, *message.Request)
    // checking
    HandleinPrePrepare(*message.PrePrepare) bool
    // nothing necessary
    HandleinPrepare(*message.Prepare) bool
    // confirming
    HandleinCommit(*message.Commit) bool
    // do for need
    HandleReqestforOldSeq(*message.RequestOldMessage) bool
    // do for need
    HandleforSequentialRequest(*message.SendOldMessage) bool
}

// Define operations among some PBFTs.
// This may be varied by different consensus protocols.
type OpInterShards interface {
    // operation inter-shards
    HandleMessageOutsidePBFT(message.MessageType, []byte) bool
}
```

Implement the interface in
./consensus_shard/pbft_all/pbftMod_interface.go
to customize different consensus operations.

supervisor > measure > measureInterface.go > ...

```go
package measure

import "blockEmulator/message"

type MeasureModule interface {
    UpdateMeasureRecord(*message.BlockInfoMsg)
    HandleExtraMessage([]byte)
    OutputMetricName() string
    OutputRecord() ([]float64, float64)
}
```

Implement the interface in
./supervisor/measure/measureInterface.go
to customize various metric measurement methods.

Go Compile & Run - Step ③： Run nodes

In Go, you can use the following commands to compile & run code:

1. Compile into **an executable file** (using go build), outputting as blockEmulator; then use the executable file as described in sections 1 to 3 previously:

```
D:\workspace_projects\block-emulator> go build -o blockEmulator.exe main.go
D:\workspace_projects\block-emulator> .\blockEmulator.exe -g --shellForExe -S 2 -N 4
```

2. Use go run to compile & run BlockEmulator directly:

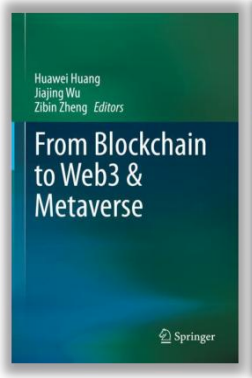
```
D:\workspace_projects\block-emulator> go run main.go -g -S 2 -N 4
```

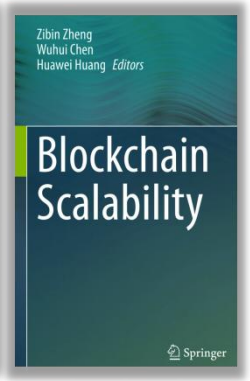
In large-scale experiments, it is recommended to use the first method (go build), which helps prevent repeated compilation in batch processing files. However, it requires ***generating different executable files for different operating systems and architectures*.

Thanks！

黄华威研究组： http://xintelligence.pro
Email： huanghw28@mail.sysu.edu.cn

感谢各位！
敬请批评指正！





Questions?

- 黄华威研究组微信号： Huang-Lab