

BlockEmulator 使用手册

(开源版本)

撰写人：叶光

指导老师：黄华威

HuangLab @ SYSU

<http://xintelligence.pro>

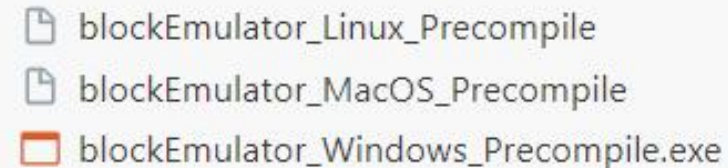
2024/07/19

本次演示：启动 blockEmulator 方式

- 2 种启动方法的区别

- 方式 1：使用提前编译好的 可执行文件

- 不需要安装 blockEmulator 所依赖的 module
 - 方便演示、体验



blockEmulator_Linux_Precompile
blockEmulator_MacOS_Precompile
blockEmulator_Windows_Precompile.exe

- 方式 2：编译 + 运行 source code

- 可以修改 网络 (IP: port)、协议 (block size, block interval) 等配置参数
 - aim to 二次开发

Outline

1. 运行代码前：事前准备
2. 运行代码时：启动 blockEmulator
3. 运行代码后：数据收集
4. 附加：安装 go 环境，修改参数运行 blockEmulator

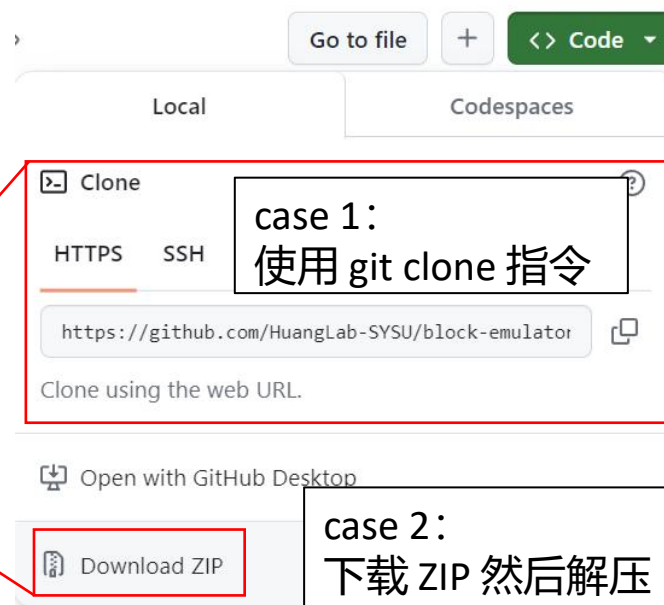
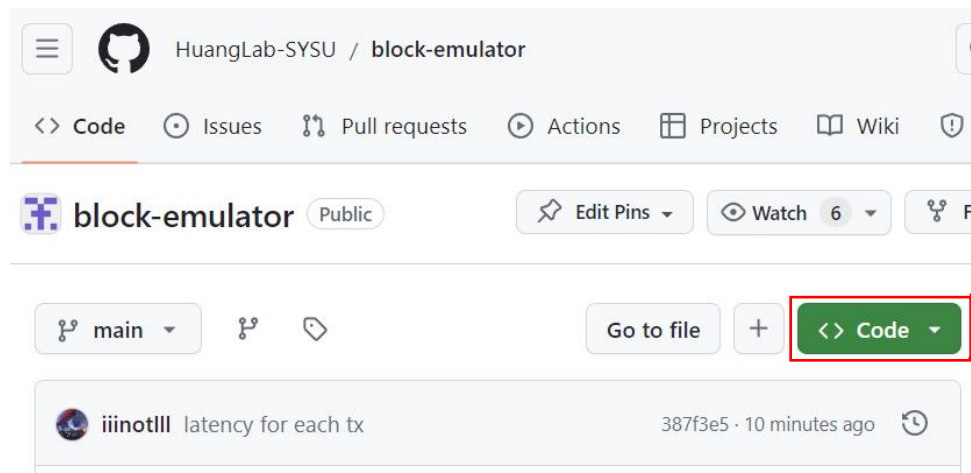
事前准备 - Step ①：拉取源代码

- 在 GitHub 上拉取 / 下载 blockEmulator 的源代码。

blockEmulator 的 GitHub 网址为：

<https://github.com/HuangLab-SYSU/block-emulator>


点击下图红圈 code 可以获取代码：



事前准备 - Step ②： 下载数据集

实验数据集来自： xblock (<https://xblock.pro/#/dataset/14>)

为了便于运行， **目前开源代码中附带了一个包含 300'000 条交易的数据集。**
上一页的“拉取源代码过程”已经获取了这个数据集。
因此， 小规模实验并不需要去 xblock 下载完整的数据集。

 go.sum	Initial commit
 main.go	.shell file generator
 selectedTx_300K.csv	new pre-provided dataset

开源代码中附带了 30万条交易数据集

Outline

1. 运行代码前：事前准备
2. 运行代码时：启动 blockEmulator
3. 运行代码后：数据收集
4. 附加：安装 go 环境，修改参数运行 blockEmulator

启动 blockEmulator - Step ①：了解参数

- blockEmulator 中的参数分为 两类：
 - 第一类参数：与共识协议的具体设计细节、网络配置相关，可以在 .go 文件中进行修改，重新编译后运行后才能生效。
 - 第二类参数：与网络规模、共识协议选择、实验结果输出目录相关，可以在命令行中直接指定，无需重新编译。

启动 blockEmulator - Step ①：了解参数

第一类参数：可在 .go 文件中改写

(下图来自 ./params/global_config.go)：

```
Block_Interval      = 5000 // The block interval
MaxBlockSize_global = 2000 // The maximum size of a block contains
InjectSpeed         = 2000 // The inject speed
TotalDataSize       = 160000 // The total data size
BatchSize           = 16000 // The batch size. The size of a batch is 'BatchSize'
BrokerNum           = 10    // The number of broker. CLPA_Broker.

DataWrite_path      = ExpDataRootDir + "/result/" // Measurement data result output path
LogWrite_path       = ExpDataRootDir + "/log"      // Log output path
DatabaseWrite_path  = ExpDataRootDir + "/database/" // database write path

SupervisorAddr = "127.0.0.1:18800" // Supervisor node's IP address
FileInput      = `./selectedTxs_300K.csv` // The dataset file address
ReconfigTimeGap = 50 // The time gap between epochs now.
```

- 出块间隔
- 每个区块包含的区块大小
- 交易注入的速度
- 用于测试的全部交易数目
- Supervisor 节点的 IP 地址
- 数据集的文件地址
- 重配置算法（如 CLPA）的执行周期

启动 blockEmulator - Step ①：了解参数

第二类参数：可在 命令行 中指定的参数
(可以在 命令行中执行 **--help** 来查看每个命令对应的功能)

比如：**.\blockEmulator_Windows_Precompile.exe --help**

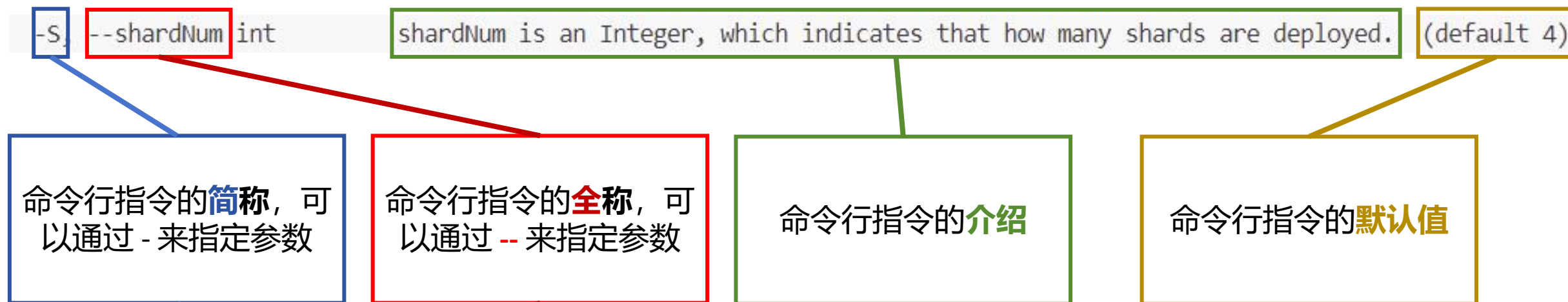
```
PS D:\workspace_projects\block-emulator> .\blockEmulator_Windows_Precompile.exe --help
Usage of D:\workspace_projects\block-emulator\blockEmulator_Windows_Precompile.exe:
-d, --dataRootDir string  dataRootDir is a string, which defines the RootDir of the experimental data, including ./log, ./record and ./re
-g, --gen                 isGen is a bool value, which indicates whether to generate a batch file
-m, --modID int           modID is an Integer, which indicates the choice ID of methods / consensuses. Value range: [0, 4), representing
-n, --nodeID int          nodeID is an Integer, which indicates the ID of this node. Value range: [0, nodeNum).
-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, shardNu
-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 execu
-c, --supervisor          isSupervisor is a bool value, which indicates whether this node is a supervisor.
pflag: help requested
```

如果是 Linux 或者 MacOS 系统，权限不够需要使用 `chmod +x {文件名}` 来提高权限。

启动 blockEmulator - Step ①：了解参数

第二类参数：在命令行中指定的参数（可以使用 --help 查看）

本页解释 --help 输出的下面这一行的意义：



也就是说，下面两行命令是等价的：

```
go run main.go -S 2 -N 4 -s 0 -n 0 -m 3
```

```
go run main.go --shardNum 2 -N 4 -s 0 -n 0 -m 3
```

启动 blockEmulator - Step ②：启动

单节点启动：

blockEmulator 可以通过这样一条命令来启动一个节点：

```
blockEmulator_Windows_Precompile.exe -n 1 -N 4 -s 0 -S 4 -m 3
```

这条命令启动了这样的一个节点：

- 网络规模：分片总数为 4，每个分片 4 个节点
- 该节点是：序号为 0 的分片内的序号为 1 的节点
- 该节点使用：-m 3 指定的 “Relay” 方式来处理跨分片交易

在上述的网络规模下，一共有 4 个分片，每个分片 4 个节点，所以需要启动 16 个节点。这样的**逐一启动很麻烦**。因此，blockEmulator 提供了**批处理启动**的脚本文件。

启动 blockEmulator - Step ②：启动

批处理生成：

-- help 指令会输出关于 -g 这条指令的信息：

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

只需要在命令行中加上“-g”，即可产生批处理文件，该批处理文件能够**批量**地启动节点。（如果是可执行文件执行该命令，需要再加上“--shellForExe”，来生成针对 exe 的批处理文件）

比如：

```
go run main.go -g -S 2 -N 4 -m 1  
.\blockEmulator_Windows_Precompile.exe -g --shellForExe -S 2 -N 4 -m 1
```

启动 blockEmulator - Step ②：启动

- 执行 生成批处理文件 命令的结果：

在 blockEmulator 目录下执行“**生成批处理文件**”的命令行指令后，以下面这条指令为例：

```
.\blockEmulator_Windows_Precompile.exe -g --shellForExe -S 2 -N 4 -m 1
```

会在 block-emulator 根目录下，会生成一个 .bat 文件①。

-m 1 (using CLPA)
-m 0 (using BrokerChain)

A screenshot of a file explorer window showing a single file named "WinExe_bat_shardNum=2_NodeNum=4_mod=CLPA.bat". The file icon is a small document with a red 'x' in the top left corner, typical of Windows batch files.

① 如果是 Linux 或者 MacOS 系统，则会生成 .sh 文件

权限不够需要使用 `chmod +x {文件名}` 来提高权限。

```
~/block-emulator$ chmod +x blockEmulator_Linux_Precompile
~/block-emulator$ ./blockEmulator_Linux_Precompile -g --shellForExe -S 2 -N 4 -m 1
```

启动 blockEmulator - Step ②: 启动

- 生成批处理文件的内容:

查看 `WinExe_bat_shardNum=2_NodeNum=4_mod=Relay.bat` 的内容。

Sequence matters. In code, using sleep for seconds.

WinExe_bat_shardNum=2_NodeNum=4_mod=CLPA.bat

```
1 start cmd /k blockEmulator_Windows_Precompile.exe -n 1 -N 4 -s 0 -S 2 -m 1 -d expTest
2
3 start cmd /k blockEmulator_Windows_Precompile.exe -n 1 -N 4 -s 1 -S 2 -m 1 -d expTest
4
5 start cmd /k blockEmulator_Windows_Precompile.exe -n 2 -N 4 -s 0 -S 2 -m 1 -d expTest
6
7 start cmd /k blockEmulator_Windows_Precompile.exe -n 2 -N 4 -s 1 -S 2 -m 1 -d expTest
8
9 start cmd /k blockEmulator_Windows_Precompile.exe -n 3 -N 4 -s 0 -S 2 -m 1 -d expTest
10
11 start cmd /k blockEmulator_Windows_Precompile.exe -n 3 -N 4 -s 1 -S 2 -m 1 -d expTest
12
13 start cmd /k blockEmulator_Windows_Precompile.exe -n 0 -N 4 -s 0 -S 2 -m 1 -d expTest
14
15 start cmd /k blockEmulator_Windows_Precompile.exe -n 0 -N 4 -s 1 -S 2 -m 1 -d expTest
16
17 start cmd /k blockEmulator_Windows_Precompile.exe -c -N 4 -S 2 -m 1 -d expTest
18
```

启动 **序号为 0** 的分片中的**非 leader** 节点。

启动 **序号为 1** 的分片中的**非 leader** 节点。

分别启动两个分片中各自的 **leader** 节点。

启动 **supervisor** 节点, 由该节点注入交易。

启动 blockEmulator - Step ②：启动

- **启动.bat/.sh 文件：**

.bat 文件可以在 Windows 系统下“**双击**”运行或使用 **命令行** 运行。

.sh 文件可以在 Linux 或 MacOS 下，使用**命令行**运行。

例子：Linux 或 MacOS **命令行**启动 .sh 文件：

在 blockEmulator 目录下输入：

```
sh bat_shardNum=2_NodeNum=4_mod=CLPA.sh
```

如果提示文件权限不够，需要 chmod 提高权限：

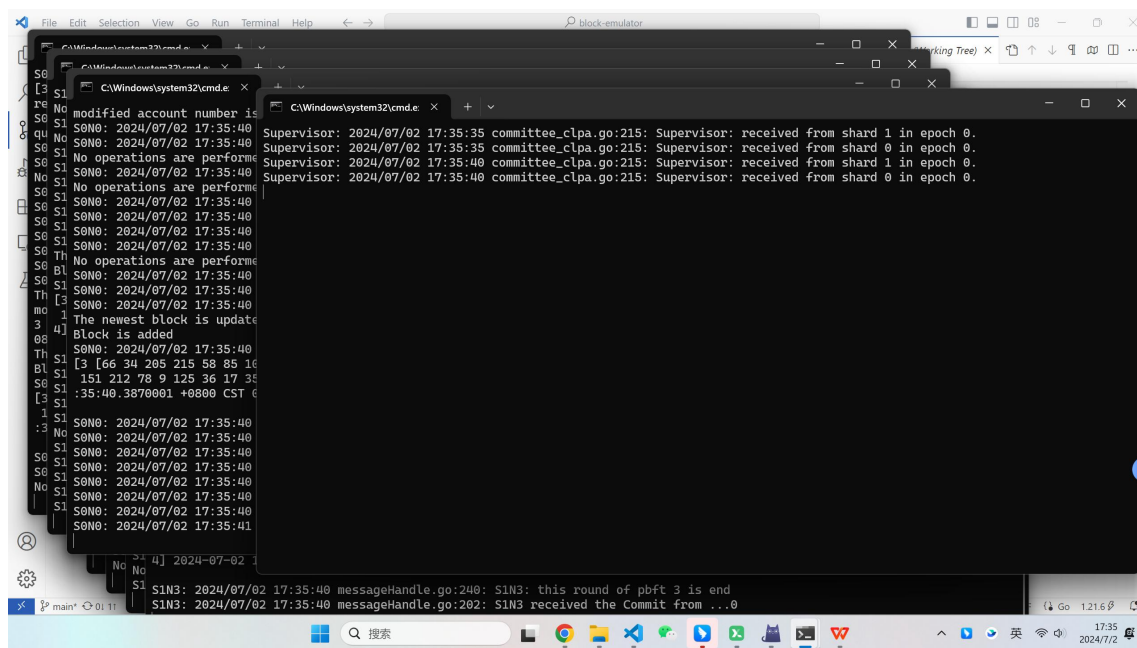
```
chmod +x bat_shardNum=2_NodeNum=4_mod=CLPA.sh
```


启动 blockEmulator - Step ③：运行中

- **blockEmulator 运行**

在 blockEmulator 运行时，用户不需要进行操作，等待结果即可。

(如下图，如果使用的操作系统为 windows，且通过 双击 .bat 文件启动批量节点，那么便会弹出若干个终端。每一个终端对应着一个节点。)



启动 blockEmulator - Step ③: log 输出

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]

启动 blockEmulator - Step ③: log 输出

Ending

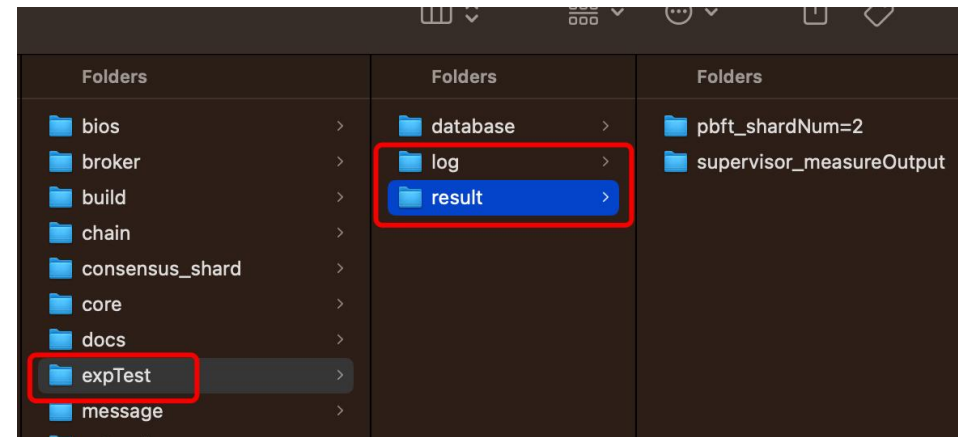
Supervisor: 2024/07/18 10:37:44 supervisor.go:223: Closing...
Supervisor: 2024/07/18 10:37:44 supervisor.go:225: Average_TPS
SONO: 2024/07/18 10:37:44 messageHandle.go:61: SONO get stopSignal in Propose Routine, now stop...
Supervisor: 2024/07/18 10:37:44 supervisor.go:226: [587.7339655544023 699.2846784883424 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



启动 blockEmulator - Step ③：运行中

- **blockEmulator 运行失败，如果端口被上一次运行的 blockEmulator 占用**
 - linux / macos : `killall -9 {blockEmulator_MacOS_Precompile}`
 - windows : 关闭所有启动的终端。

Outline

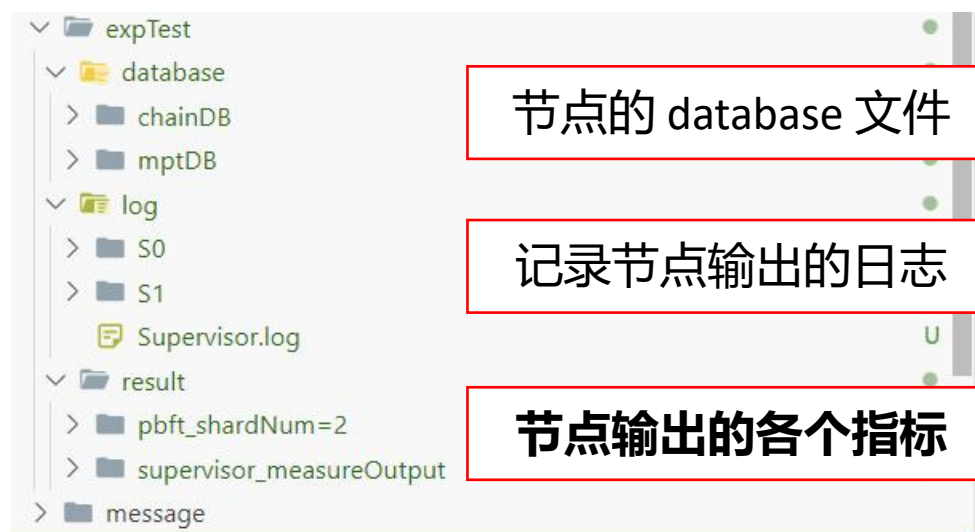
1. 运行代码前：事前准备
2. 运行代码时：启动 blockEmulator
3. 运行代码后：数据收集
4. 附加：安装 go 环境，修改参数运行 blockEmulator

数据收集 - Step ①：查看被更新的文件夹

- blockEmulator 运行完成后，可以看到 三个文件夹 被更新了。

它们分别是： **./expTest** 目录下的 ./log, ./record, ./result

./expTest 可以在命令行中使用 **-d** 指定，从而每次实验结果可以储存在不同的目录下。



数据收集 - Step ②：查看 ./result

- 三个文件夹中， ./result 记录节点输出的指标，这一般是我们最重视的部分。

result

pbft_shardNum=2

Shard02.csv

Shard12.csv

由每个分片的 leader 节点 输出的测试指标。
其中包含：每个分片内的交易池大小变化、
每个区块处理的交易数目等内容。

supervisor_measureOutput

Average_TPS.csv

CrossTransaction_ratio.csv

Transaction_Confirm_Latency.csv

Tx_Details.csv

Tx_number.csv

由 supervisor 节点 输出的测试指标。其中包含：
平均 TPS，跨分片交易比率，交易确认时延，
交易总数等内容。此部分可以
在 ./supervisor/measure 文件夹下参考现有代
码来DIY。

数据收集 - Step ②：查看 ./result

- 查看 每个分片 leader 输出的 实验结果：

以 `.\expTest\result\pbft_shardNum=2\Shard02.csv` 为例：

	A	B	C	D	E	F	G	H	I	J	K
1	Block Height	EpochID of TxPool	Size of all 1	# of Relay	# of Relay	Time Stamp	Time Stamp	SUM of cor	SUM of cor	SUM of cor	
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

指标名称

指标数值

blockHeight 为 1 时，区块内没有交易，这是因为 Supervisor 此时尚未开始注入交易。

数据收集 - Step ②：查看 ./result

- 查看 Supervisor 输出的 实验结果 1：

以 `.\expTest\result\supervisor_measureOutput\Average_TPS.csv` 为例：

	A	B	C	D	E	F	G	H
1	EpochID	Total tx	Normal tx	Relay1 tx	Relay2 tx	Epoch start	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

指标名称

指标数值

数据收集 - Step ②：查看 ./result

- 查看 Supervisor 输出的 实验结果 2：

以 `.\expTest\result\supervisor_measureOutput\Tx_Details.csv` 为例：

#	A	B	C	D	E	F	G	H	I
1	TxHash (B)	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

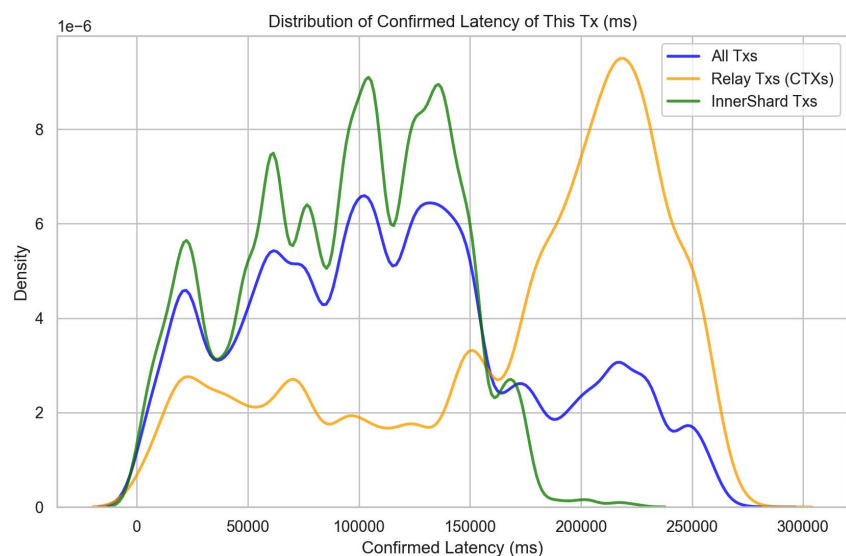
指标名称

指标数值

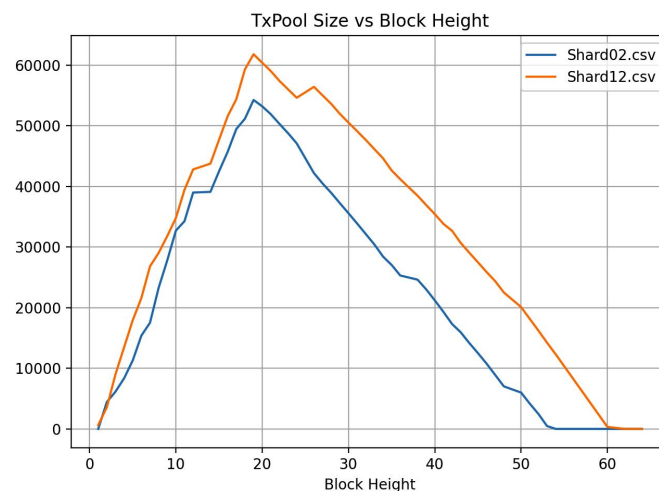
由于我们刚刚使用的是 CLPA 算法，所以 Broker 交易相关的这两项均为空。

数据收集 - Step ③：使用数据绘制图像

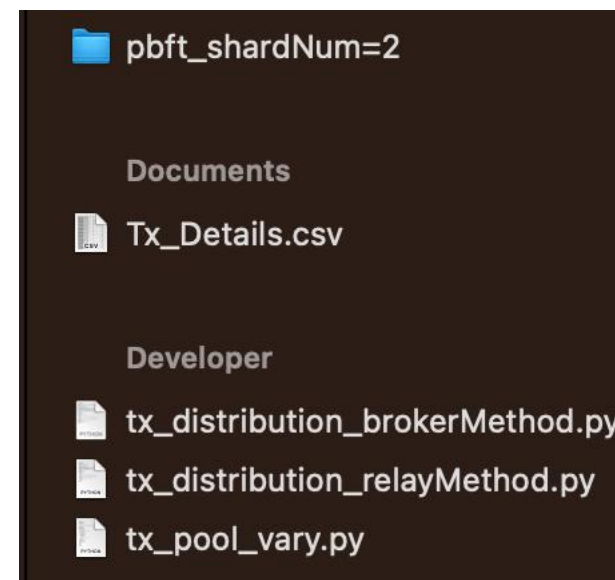
使用./result 内的数据来画图（画图代码位于 ./figurePlot ）



`python tx_distribution_relayMethod.py`



`python tx_pool_vary.py`



Outline

1. 运行代码前：事前准备
2. 运行代码时：启动 blockEmulator
3. 运行代码后：数据收集
4. 附加：安装 go 环境，修改参数运行 blockEmulator

Go 编译运行 - Step ①：配置 Go 环境

- 编译&运行 blockEmulator 需要安装 Go

Go 的官方下载网址为：<https://go.dev/dl/>
进入上面的网站后，下载安装即可。

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 disable 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 编译运行 - Step ①：配置 Go 环境

在 blockEmulator 目录下，执行 **go mod tidy**，下载运行代码所依赖的 module（具体需要哪些 modules 可以见 ./go.mod 文件）。

```
go.mod
Reset go.mod diagnostics | Run go mod tidy | Create vendor directory
1 module blockEmulator
2
3 go 1.19
4
5 Check for upgrades | Upgrade transitive dependencies | Upgrade direct dependencies
6 require (
7     github.com/boltdb/bolt v1.3.1
8     github.com/ethereum/go-ethereum v1.11.6
9     github.com/spf13/pflag v1.0.5
10 )
```

./go.mod 文件里指明了所依赖的 modules

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

执行 **go mod tidy** 命令，自动下载安装所依赖的 modules

依赖文件的国内镜像下载：

go env -w GOPROXY=<https://goproxy.cn,direct>

go env -w GO111MODULE=on

Go 编译运行 - Step ②：修改文件内参数

现在，我们可以修改 blockEmulator 中写在 ./go 文件里的参数了。

```
func initConfig(nid, nnm, sid, snm uint64) *params.ChainConfig {  
    params.ShardNum = int(snm)  
    for i := uint64(0); i < snm; i++ {  
        if _, ok := params.IPmap_nodeTable[i]; !ok {  
            params.IPmap_nodeTable[i] = make(map[uint64]string)  
        }  
        for j := uint64(0); j < nnm; j++ {  
            params.IPmap_nodeTable[i][j] = "127.0.0.1:" + strconv.Itoa(28800+int(i)*100+int(j))  
        }  
    }  
}
```

.\build\build.go 文件中可以修改每个节点对应的 IP 表。
(如果需要多机部署 blockEmulator，就需要修改 IP 表。)

```
var (  
    Block_Interval      = 5000 // The time interval for generating a new block  
    MaxBlockSize_global = 2000 // The maximum number of transactions a block contains  
    InjectSpeed         = 2000 // The speed of transaction injection  
    TotalDataSize       = 160000 // The total number of txs to be injected  
    BatchSize           = 16000 // The supervisor read a batch of txs then send them. The size of a batch is 'BatchSize'  
    BrokerNum           = 10    // The # of Broker accounts used in Broker / CLPA_Broker.  
  
    DataWrite_path      = ExpDataRootDir + "/result/" // Measurement data result output path  
    LogWrite_path       = ExpDataRootDir + "/log"     // Log output path  
    DatabaseWrite_path  = ExpDataRootDir + "/database/" // database write path  
  
    SupervisorAddr = "127.0.0.1:18800" // Supervisor ip address  
    FileInput      = `./selectedTxs_300K.csv` // The raw BlockTransaction data path  
  
    ReconfigTimeGap = 50 // The time gap between epochs. This variable is only used in CLPA / CLPA_Broker now.  
)
```

.\params\global_config.go 文件中可以修改 PBFT 共识相关的一些参数。

Go 环境下编译运行 - Step ③：编译&运行

在 Go 语言中，编译&运行可以使用如下命令：

1. 编译为可执行文件（使用 go build），输出为 blockEmulator 文件；然后使用可执行文件操作（和之前 1~3 部分介绍的可执行文件操作一样）：

```
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 -m 1
```

包含 main() 函数，是程序入口点

2. 直接编译&运行（使用 go run）：

```
D:\workspace_projects\block-emulator> go run main.go -g -S 2 -N 4 -m 1
```

在大规模实验时，推荐使用第一种方法，能够防止在批处理运行文件中重复编译，但是需要为不同操作系统和架构生成不同的可执行文件。

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

