

# Technical Document

## Efficient Concurrent Transaction Execution Framework for Blockchains

Manaswini Piduguralla, Saheli Chakraborty, Parwat Singh Anjana, Sathya Peri

June 1, 2023

### 1 Introduction

The purpose of this document is to explain how to install, run, and test the Enhanced Sawtooth in Ubuntu OS. We have designed a parallel direct acyclic graph (DAG) based parallel scheduler module for concurrent execution of SCTs. Using the module, which can be seamlessly integrated into the blockchain framework, parallel transactions can be executed more efficiently and the speed of transactions can be increased. The dependencies among a block's transactions are represented through a concurrent DAG data structure that assists with throughput optimization. We have created a DAG scheduler module that can be incorporated into blockchain platforms for concurrent execution with ease.

For evaluation, our framework is implemented in Hyperledger Sawtooth V1.2.6. Our artifact includes two implementations of the DAG scheduler, one with a linked list implementation and one with an adjacency matrix implementation. We have tested our DAG modules with Sawtooth's in-built transaction families and publicly available transaction families. We have utilized the following transaction families for evaluation:

1. Intkey transaction family
2. SimpleWallet transaction family
3. Voting transaction family
4. Insurance transaction family

### 2 Getting Started Guide

#### 2.1 Installation of Sawtooth Blockchain node

We have implemented and deployed our proposed framework for efficient and concurrent transaction execution for blockchains in Sawtooth 1.2.6. Follow the steps to setup a Sawtooth node.

[Link for artifact](#)

##### 2.1.1 Prerequisites

1. Compatible operating system:  
Sawtooth 1.2.6 is officially supported on Linux distributions such as Ubuntu and Fedora.
2. Docker is installed in your system:  
Sawtooth utilizes Docker containers for component isolation. Steps for installation are in Section 2.1.2

### 3. Python 3.6 or higher:

The Sawtooth node is developed with a combination of Python and Rust. Docker node installs Rust during setup.

### 4. Commands:

```
$ sudo apt install build-essential
$ sudo apt install apt-utils
$ sudo apt-get install python3-pip
$ pip3 install secp256k1
$ pip3 install pyformance
```

#### 2.1.2 Docker Installation steps

Sawtooth development environment requires Docker Engine and Docker Compose. We have observed that Sawtooth node works best with the below versions of docker engine and docker compose:

1. docker Docker version 20.10.17
2. docker compose : docker-compose version 1.25.0

One can check their current available version of the packages using the commands:

```
$ docker-compose --version
$ docker --version
```

**Install Docker Engine on Ubuntu :** The first command prints the available docker engine version. select the version 20.10.17 using version string in the second command as shown below.

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sudo chmod +x ./get-docker.sh
$ sudo ./get-docker.sh
$ apt-cache madison docker-ce | awk '{ print $3 }'
$ VERSION_STRING=5:20.10.17~3-0~ubuntu-bionic
$ sudo apt-get install docker-ce=$VERSION_STRING \
    docker-ce-cli=$VERSION_STRING containerd.io \
    docker-buildx-plugin docker-compose-plugin
```

If unable to locate the specific docker version use:

```
$ sudo snap install docker
```

**Install Docker Compose on Ubuntu :**

```
$ mkdir ~/.docker
$ export DOCKER_CONFIG=${HOME}/.docker
$ sudo mkdir -p $DOCKER_CONFIG/cli-plugins
$ curl -L \
    "https://github.com/docker/compose/releases/download/1.25.0/docker-compose-$(uname \
        -s)-$(uname -m)" -o \
        $DOCKER_CONFIG/cli-plugins/docker-compose
$ chmod +x $DOCKER_CONFIG/cli-plugins/docker-compose
$ export PATH=~/.docker/cli-plugins/:$PATH
```

For further information the guide for installation of docker and docker compose are present in the below links.

1. Docker Engine
2. Docker compose

### 2.1.3 Building and Run a Sawtooth node

The Sawtooth node is created using dockers. Follow the steps to build and run the node in the local system:

#### Step 1:

Download and extract the Enhanced Sawtooth in the local system.

```
$ unzip Enhance_Sawtooth.zip
```

#### Step 2:

Open the Enhanced\_Sawtooth location in the terminal and build the node

```
$ cd Enhance_Sawtooth
$ sudo docker-compose build
```

The build command download all the required packages and docker image of the Sawtooth node. We have observed that on average the build command takes 30 to 40 minutes depending on the internet connectivity. The build completion message will be as Figure 1.

```
---> Running in 72348d4e1455
Removing intermediate container 72348d4e1455
---> ad458268ed00
Step 9/11 : ENV PATH=$PATH:/project/sawtooth-core/bin
---> Running in 9a421d7223e3
Removing intermediate container 9a421d7223e3
---> 1ce9c92d6101
Step 10/11 : WORKDIR /project/sawtooth-core
---> Running in 8793567b3c9c
Removing intermediate container 8793567b3c9c
---> 9db92777d7b5
Step 11/11 : CMD echo "\033[0;32m--- Building rest_api ---\n\033[0m" && bin/p
rotogen && cd rest_api && python3 setup.py clean --all && python3 setup.py
build
---> Running in 8f422a772ff7
Removing intermediate container 8f422a772ff7
---> a07de2ab47a3

Successfully built a07de2ab47a3
Successfully tagged sawtooth-rest-api-local:latest
```

Figure 1: Build completion.

Note: After installing Rust the next step takes a little time to start and we request the users to wait for few minutes.

If network timeout error occurs, it indicates either the github server is busy or the local network speed is low. Restarting the build command after sometime is the only solution.

```
sawtooth-rest-api-local | [2023-05-24 06:25:00-125] INFO [rest_api] Starting REST API on rest-api:8090
sawtooth-validator-local | warning: spurious network error (2 tries remaining): failed to connect to github.com: Connection timed out; class=Os (2)
sawtooth-rest-api-local | warning: spurious network error (2 tries remaining): failed to connect to github.com: Connection timed out; class=Os (2)
sawtooth-validator-local | warning: spurious network error (1 tries remaining): failed to connect to github.com: Connection timed out; class=Os (2)
sawtooth-rest-api-local | warning: spurious network error (1 tries remaining): failed to connect to github.com: Connection timed out; class=Os (2)
sawtooth-devmode-engine-rust | Error: [devmode_engine_rust] ReceivedError: TimeoutError
sawtooth-devmode-engine-rust | exited with code 1
sawtooth-validator-local | error: Unable to update registry 'crates.io'
sawtooth-validator-local | Caused by:
sawtooth-validator-local | failed to fetch 'https://github.com/rust-lang/crates.io-index'
```

Figure 2: Network Error.

#### Step 3:

To start the Sawtooth node :

```
$ sudo docker-compose up
```

We have observed that it typically takes 2 minutes for the Sawtooth node to start. Creation of genesis block i.e., block 0, indicates that the node has started correctly. The log can be observed in Figure 3. If no scheduler is selected **"ModuleNotFoundError: No module named"** occurs.

```
006ebcb8a8a9fc5382ed2dc4cac840ef4b92b8c666677444b79f3d619adc1f7df281cec1deeb
8161018555134b23210f887b30c5e00fb239bf6cfd1df4690b, block_num: 0, state_root
_hash: 2970dca23b6aaaf210d970bc591c827c3ee482ec242eb51ab9af18748c56a594, pre
vious_block_id: 0000000000000000)
sawtooth-validator-local | [2023-05-13 08:02:54.092 INFO      ffl] [src/journ
al/publisher.rs: 172] Now building on top of block, Block(id: 18006ebcb8a8a9
fc5382ed2dc4cac840ef4b92b8c666677444b79f3d619adc1f7df281cec1deeb816101855513
4b23210f887b30c5e00fb239bf6cfd1df4690b, block_num: 0, state_root_hash: 2970d
ca23b6aaaf210d970bc591c827c3ee482ec242eb51ab9af18748c56a594, previous_block_
id: 0000000000000000)
sawtooth-xo-tp-python | [2023-05-13 08:02:54.116 INFO      core] register att
empt: OK
sawtooth-validator-local | [2023-05-13 08:02:54.167 INFO      handlers] Conse
nsus engine registered: Devmode 0.1 (additional protocols: [])
sawtooth-validator-local | [2023-05-13 08:02:54.170 INFO      proxy] Consensu
s engine activated: Devmode 0.1
sawtooth-devmode-engine-rust | INFO      | devmode_engine_rust: | Wait time: 0
sawtooth-devmode-engine-rust | INFO      | devmode_engine_rust: | Timer expired
-- publishing block
sawtooth-intkey-tp-python | [2023-05-13 08:02:54.316 INFO      core] register
```

Figure 3: Sawtooth node set up

In the current version the tree scheduler is setup as it is the default scheduler. The steps to change the scheduler are presented in Section 2.1.4

#### Step 4:

To stop the Sawtooth node :

```
$ sudo docker-compose down
```

```
manaswani@sudarshan:~/sawtooth/sawtooth-exper$ docker-compose down
Stopping sawtooth-shell-local      ... done
Stopping sawtooth-intkey-tp-python ... done
Stopping sawtooth-settings-tp-local ... done
Stopping sawtooth-rest-api-local   ... done
Stopping sawtooth-xo-tp-python     ... done
Stopping sawtooth-devmode-engine-rust ... done
Stopping sawtooth-validator-local  ... done
Removing sawtooth-shell-local      ... done
Removing sawtooth-intkey-tp-python ... done
Removing sawtooth-settings-tp-local ... done
Removing sawtooth-rest-api-local   ... done
Removing sawtooth-xo-tp-python     ... done
Removing sawtooth-devmode-engine-rust ... done
Removing sawtooth-validator-local  ... done
Removing network sawtooth-exper_default
```

Figure 4: Sawtooth node stops

### 2.1.4 Selecting each scheduler

There are four types schedulers present our artifact.

1. Serial scheduler
2. Tree scheduler
3. Linked List DAG scheduler
4. Adjacency Matrix DAG scheduler

Before the start running the node, one needs to follow different steps to select different schedulers.

**Serial scheduler:**

1. Open "docker-compose.yaml" file present inside in folder "Enhanced\_Sawtooth"
2. change line 106 to

```
sawtooth-validator --scheduler serial -vv \
```

make sure the indentation is followed correctly.

**Tree scheduler:**

1. Open "docker-compose.yaml" file present inside in folder "Enhanced\_Sawtooth"
  2. change line 106 to
- ```
sawtooth-validator --scheduler parallel -vv \
```
3. copy file "scheduler\_parallel\_tree.py" content to "scheduler\_parallel.py" in location "Enhanced\_Sawtooth/validator/sawtooth\_validator/execution"

**Linked List DAG scheduler:**

1. Open "docker-compose.yaml" file present inside in folder "Enhanced\_Sawtooth"
  2. change line 106 to
- ```
sawtooth-validator --scheduler parallel -vv \
```
3. Navigate to folder location "Enhanced\_Sawtooth/validator/sawtooth\_validator/execution"
  4. Copy contents of "scheduler\_parallel\_LL\_DAG" to "scheduler\_parallel"
  5. Copy the contents of the folder "Enhanced\_Sawtooth/validator/LL\_DAG" to "Enhanced\_Sawtooth/validator/DAG"

**Adjacency Matrix DAG scheduler:**

1. Open "docker-compose.yaml" file present inside in folder "Enhanced\_Sawtooth"
  2. change line 106 to
- ```
sawtooth-validator --scheduler parallel -vv \
```
3. Navigate to folder location "Enhanced\_Sawtooth/validator/sawtooth\_validator/execution"
  4. Copy contents of "scheduler\_parallel\_ADJ\_DAG" to "scheduler\_parallel"
  5. Copy the contents of the folder "Enhanced\_Sawtooth/validator/ADJ\_DAG" to "Enhanced\_Sawtooth/validator/DAG"

### 3 Step-by-Step Instructions

In Section 2 we have seen the installation and set-up of the Sawtooth node in the local system. Following this the instructions to choose a scheduler among the four options is detailed. Now we will walk you through running the experiments for each transaction family. As we have mentioned in Section 1, there are four transaction families incorporated in this artifact. The execution time for each block is logged in to the file "timing\_computation.txt". store in location "Enhanced\_Sawtooth/validator" For each block added to the blockchain two execution times are presented one while block creation and one for block validation. We have provided the test cases of the experiments which we have depicted in the graphs in Section 4.2 of our paper.

### 3.1 Specifications of our experimental setup

1. Operating System: Ubuntu 18.04
2. Model name = Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz
3. RAM = 32 GB
4. Cores: 28
5. Threads per core: 2

### 3.2 Simplewallet Transaction Family

Simplewallet transaction family implements wallet operations. Some key operations provided by this transaction family are:

1. Keygen : This creates the account for an account holder which is a prerequisite for executing the rest of the following commands
2. Deposit : This deposits an amount in the wallet account of the person
3. Balance : This retrieves the balance of a particular wallet account holder
4. Withdraw: This withdraws an amount from the wallet account of the person
5. Transfer: This performs transfer of money from one wallet account to another

We have stored sample test cases for Simplewallet transaction family in the folder "Enhanced.Sawtooth/simplewallet\_files". To run please follow the below instructions:

#### Terminal one:

```
$ sudo docker-compose up
```

Wait for the node to create genesis block before starting the transaction family processor and clients.

#### Terminal two (Client):

```
$ sudo docker exec -it sawtooth-shell-local bash
$ sawtooth keygen client
$ cd pyclient
```

#### Terminal three (Processor):

```
$ sudo docker exec -it sawtooth-shell-local bash
$ cd pyprocessor
$ ./simplewallet-tp
```

#### Experiment one:

1. Go to the location "Enhanced.Sawtooth/simplewallet\_files/Exp1\_CP1"
2. Copy the files keyGen.txt, iniDepo.txt and testFile.txt paste it in "Enhanced.Sawtooth/pyclient"
3. Terminal two (client):Submit the transactions by executing the code

```
$ python3 script.py
```

#### Experiment two:

1. Go to the location "Enhanced.Sawtooth/simplewallet\_files/Exp2\_CP2"
2. Now 2 boundary condition test files (200 txns, 1200 txns) are provided.
3. Select one test case to follow the below instruction.

```

39f4e47b554c90fda2dbae87b5adabd5998709946fbb2b03efd17c920593f5b5b4a2f3ac53
bb1775146a427d22a09a6fd13fd1bf0f8266287b9. Using default max occupancy: 56
sawtooth-validator-local | [2023-05-12 08:24:22.685 INFO processor_hand
lers] registered transaction processor: connection_id=328adb3e099939f4e47b5
54c90fda2dbae87b5adabd5998709946fbb2b03efd17c920593f5b5b4a2f3ac53bb1775146
a427d22a09a6fd13fd1bf0f8266287b9, family=simplewallet, version=1.0, namespa
ces=['7e2664'], max_occupancy=56
sawtooth-validator-local | [2023-05-12 08:24:29.623 DEBUG interconnect]
No response from 2a666f69a59b206b27f479e811869e54de45d3bd7bb7ff887d89a4e5cb
c148a859e9a64e8f224854f0c17e2c9776be157bb7eab09c757ca0eda1b0eb922e71fb in 1
0.006080150604248 seconds - beginning heartbeat pings.
sawtooth-validator-local | [2023-05-12 08:24:29.624 DEBUG interconnect]
No response from 76779a358bedb419d6c8bdf792f3c19a461276526a5a1fa8f0e5f7528d
7ad0d19764da6b01e3d95703b8e086b920489386d58d81f212bb8b94097334a45c1dd2 in 1
0.004326581954956 seconds - beginning heartbeat pings.

```

Figure 5: SimpleWallet Transaction Processor successfully added to the node

```

[23:52:38 DEBUG connectionpool] http://rest-api:8008 "POST /batches HTTP/1.1
" 202 183
[23:52:38 DEBUG connectionpool] http://rest-api:8008 "POST /batches HTTP/1.1
" 202 183
[23:52:38 DEBUG connectionpool] http://rest-api:8008 "POST /batches HTTP/1.1
" 202 183
Response: {
  "link": "http://rest-api:8008/batch_statuses?id=1b7326b24c70d131496b62ce9973
fac7afeed3712b37b72480a548bb0ac39e3511f38f2ac75f6ace93f61637015da3237a2b2bce54
04d0af2ae1a8e7f9320e77"
}

```

Figure 6: Simplewallet client terminal after successful batch submission

4. Copy the files keyGen.txt, iniDepo.txt and testFile.txt paste it in "Enhanced\_Sawtooth/pyclient"
5. Terminal two (client):Submit the transactions by executing the code

```
$ python3 script.py
```

### Experiment three:

1. Go to the location "Enhanced\_Sawtooth/simplewallet\_files/Exp3-CP3"
2. Now 2 boundary condition test files (dep0, dep10) are provided.
3. Select one test case to follow the below instruction.
4. Copy the files keyGen.txt, iniDepo.txt and testFile.txt paste it in "Enhanced\_Sawtooth/pyclient"
5. Terminal two (client):Submit the transactions by executing the code

```
$ python3 script.py
```

## 3.3 Voting Transaction Family

Voting Transaction family has functionalities which can help to implement a voting system in blockchain. The key functionalities provided by this are:

1. Addparty : This creates a party to vote for. A party can be created only once. Parties are identified by their names.
2. Votefor : Voters can cast vote exactly once and for exactly one party. Voters are identified by their names.
3. Listparties : This creates a list of all the parties for whom the voters can vote.
4. Listvoters : This creates a list of all the voters who have cast their vote.
5. Voteount : This commands lists down the number of votes received by specified party.

We have stored sample test cases for voting transaction family in the folder "Enhanced\_Sawtooth/voting\_files". To run please follow the below instructions:

### Terminal one:

```
$ sudo docker-compose up
```

Wait for the node to create genesis block before starting the transaction family processor and clients.

**Terminal two (Client):**

```
$ sudo docker exec -it sawtooth-shell-local bash
$ cd voting_client
```

**Terminal three (Processor):**

```
$ sudo docker exec -it sawtooth-shell-local bash
$ cd voting_tp
$ python3 tp.py
```

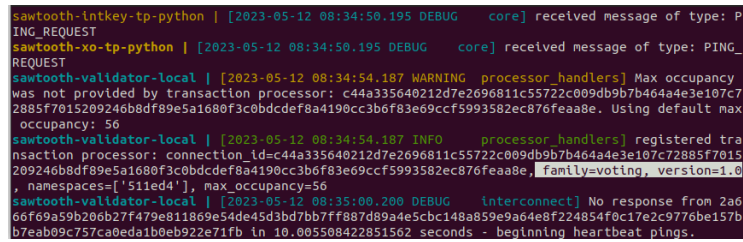
The screenshot shows a terminal window with three lines of log output. The first line is from 'sawtooth-intkey-tp-python' showing a 'PING\_REQUEST' message. The second line is from 'sawtooth-xo-tp-python' also showing a 'PING\_REQUEST' message. The third line is from 'sawtooth-validator-local' showing a 'WARNING' about 'processor\_handlers' and an 'INFO' about 'registered transaction processor' with various connection details and namespace information.

Figure 7: Voting Transaction Processor successfully added to the node

**Experiment one:**

1. Copy the content of voting\_files/Exp1-CP3/testFile.txt and paste in voting\_client/batches.txt
2. Terminal two (client): Submit the transactions by executing the code

```
$ python3 vote.py
```

**Experiment two:**

1. Now 2 boundary condition test files (200, 1200) are provided.
2. Copy the content of voting\_files/Exp2-CP2/folder/testFile.txt and paste in voting\_client/batches.txt
3. Terminal two (client): Submit the transactions by executing the code

```
$ python3 vote.py
```

**Experiment three:**

1. Now 2 boundary condition test files (dep0, dep10) are provided.
2. Copy the content of voting\_files/Exp3-CP1/folder/testFile.txt and paste in voting\_client/batches.txt
3. Terminal two (client): Submit the transactions by executing the code

```
$ python3 vote.py
```

### 3.4 Intkey Transaction Family

Intkey is a simple transaction family developed for mainly testing purpose. The functionalities in this transaction family are:

1. Set : It creates a key and sets it to a particular value.
2. Increment : It increments the value of a key by the specified value.
3. Decrement : It decrements the value of a key by the specified value.



4. Show : This shows the current value of the specified key
5. List: It displays the value of all keys.

We have stored sample test cases for Intkey transaction family in the folder "Enhanced\_Sawtooth/intkey\_files". To run please follow the below instructions:

**Terminal one:**

```
$ sudo docker-compose up
```

Wait for the node to create genesis block before starting the transaction family processor and clients.

**Terminal two (Client):**

```
$ sudo docker exec -it sawtooth-shell-local bash
$ cd intkey_client
```

**Terminal three (Processor):**

```
$ sudo docker exec -it sawtooth-shell-local bash
$ python3 ./intkey_processor/main.py
```

**Experiment one:**

1. Now 2 boundary condition test files (200, 1200) are provided.
2. Terminal two (client): Submit the transactions by executing the code
3. For initializing keys to execute 200 transactions
 

```
$ sawtooth batch submit -f batches_200init.intkey --url http://rest-api:8008
```
4. for submitting 200 txns execution we can do
 

```
$ sawtooth batch submit -f batches200.intkey --url http://rest-api:8008
```
5. For initializing keys to execute 1200 transactions
 

```
$ sawtooth batch submit -f batches_1200init.intkey --url http://rest-api:8008
```
6. for submitting 1200 txns execution we can do
 

```
$ sawtooth batch submit -f batches1200.intkey --url http://rest-api:8008
```

### 3.5 Insurance Transaction Family

Insurance Transaction family has functionalities which can help maintain the records of all the insurance holders in the system. The key functionalities provided by this are:

1. adduser: This stores the user details in the database.
2. view: We can lookup the user details based on their ID

We have stored sample test cases for Insurance transaction family in the folder "Enhanced\_Sawtooth/insurance\_files". To run please follow the below instructions:

**Terminal one:**

```
$ sudo docker-compose up
```

Wait for the node to create genesis block before starting the transaction family processor and clients.

**Terminal two (Client):**

```
$ sudo docker exec -it sawtooth-shell-local bash
$ cd Ins_client
```

#### Terminal three (Processor):

```
$ sudo docker exec -it sawtooth-shell-local bash
$ cd Ins_processor/insurance-handler
$ apt install nodejs
please Y when prompted in the terminal
$ node Processor.js
```

Figure 8: Insurance Transaction Processor successfully added to the node

#### Experiment One:

1. Open Ins\_client/batches.txt
2. Now 2 boundary condition test files (200, 1200) are provided.
3. Copy the content of insurance\_files/Exp2.CP1/folder/testFile.txt and paste in batches.txt
4. Terminal two (client):Submit the transactions by executing the code

```
$ node insuranceLogic.js
```

We have provided the two extreme cases for all the graphs we have depicted in the paper. The detailed test cases are present in the folder "Enhanced.Sawtooth/Complete.Experiments". As we have mentioned earlier the execution time is appended in the "Enhanced.Sawtooth/validator/timing.computation.txt".

## 4 Instructions for performing the experiments

### 4.1 Simplewallet Transaction Family

We have developed scripts to automate the execution of experiments and calculate both the execution time and data creation time. The validation time of the smart validator is calculated in the ADJ.DAG scheduler implementation. Detailed instructions for conducting the experiments are provided.

#### Terminal one:

```
$ sudo docker-compose up
```

Wait for the node to create a genesis block before starting the transaction family processor and clients.

#### Terminal two (Client):

```
$ sudo docker exec -it sawtooth-shell-local bash
$ sawtooth keygen client
$ cd pyclient
```

### Terminal three (Processor):

```
$ sudo docker exec -it sawtooth-shell-local bash
$ cd pyprocessor
$ ./simplewallet-tp
```



```
[23:52:38 DEBUG connectionpool] http://rest-api:8008 "POST /batches HTTP/1.1"
" 202 183
[23:52:38 DEBUG connectionpool] http://rest-api:8008 "POST /batches HTTP/1.1"
" 202 183
[23:52:38 DEBUG connectionpool] http://rest-api:8008 "POST /batches HTTP/1.1"
" 202 183
[23:52:38 DEBUG connectionpool] http://rest-api:8008 "POST /batches HTTP/1.1"
" 202 183
[23:52:38 DEBUG connectionpool] http://rest-api:8008 "POST /batches HTTP/1.1"
" 202 183
[23:52:38 DEBUG connectionpool] http://rest-api:8008 "POST /batches HTTP/1.1"
" 202 183
Response: {
  "link": "http://rest-api:8008/batch_statuses?id=1b7326b24c70d131496b62ce9973
fac7afeed3712b37b72480a548bb0ac39e3511f38f2ac75f6ace93f61637015da3237a2b2bce54
04d0af2ae1a8e7f9320e77"
}
ADJ DAG data struction creation time: 0.06700873374938965
smart validator verification time: 0.039049386978149414
Execution Time : 35.483123540878296
```

Figure 9: SimpleWallet client terminal with total execution time and data structure creation time

### automated script explanation:

1. The first step in the code is to execute the “keygen“ function. This function is responsible for generating the necessary cryptographic keys required for the execution of the subsequent steps.
2. The code renames the file “iniDepo“ to “batches.txt“ and then proceeds to execute it. This file contains initial deposit data.
3. After executing the “batches.txt“ file, the code removes the timing information from the file named “timing\_computation.txt“. As the time for initial deposit batch execution is not required for the experiment.
4. The code contains a hard-coded value of 10 for the number of blocks. This means that the subsequent steps will be executed for a fixed number of blocks, which is 10 in this case. If you want to change the number of blocks, you can modify the value assigned to the variable “numBlocks“ in the code.
5. The code proceeds to execute each block one by one. For each block, it performs batch submission and records the execution time in the “timing\_computation.txt“ file.
6. Once all the executions for the 10 blocks are completed, the code extracts the execution times recorded in the “timing\_computation.txt“ file. It then calculates the sum of the execution times for the 20 blocks. Since the blocks are executed twice the total time is for 20 executions.
7. Total time for the data structure created for the smart validator plots.

### Experiment one:

1. Go to the location “Enhanced\_Sawtooth/Complete\_Experiments/simplewallet“

2. select one experiment and one conflict percentage.
3. Copy the files keyGen.txt, iniDepo.txt and testFile.txt paste it in “Enhanced\_Sawtooth/pyclient”
4. Modify the value of “numBlocks” in the script.py to 5.
5. Terminal two (client): Submit the transactions by executing the code

```
$ python3 script.py
```

To generate the data points for the plots displayed in the paper, the same batch is submitted multiple times, ranging from 10 to 50 times. The script is executed to obtain the execution time for each run of the same batch. In this particular case, running the script provided the execution time for 10 runs of the same batch.

#### Experiment two:

1. Go to the location “Enhanced\_Sawtooth/Complete\_Experiments/simplewallet”
2. select one experiment and one conflict percentage.
3. Test files (200 txns to 1200 txns) are provided.
4. Select one test case to follow the below instruction.
5. Copy the files keyGen.txt, iniDepo.txt and testFile.txt paste it in “Enhanced\_Sawtooth/pyclient”
6. Modify the value of “numBlocks” in the script.py to 10.
7. Terminal two (client): Submit the transactions by executing the code

```
$ python3 script.py
```

To generate the data points for the plots presented in the paper, batches with transaction sizes ranging from 200 transactions to 1200 transactions are submitted a total of 20 times for each batch size. The script is executed to measure the execution time for each run of the same batch. In this specific scenario, running the script provided the execution time for 20 runs of each batch, ensuring a robust dataset for analysis and plotting.

#### Experiment three:

1. Go to the location “Enhanced\_Sawtooth/Complete\_Experiments/simplewallet”
2. select one experiment and one conflict percentage.
3. Test files (dep0 to dep10) are provided.
4. Select one test case to follow the below instruction.
5. Copy the files keyGen.txt, iniDepo.txt and testFile.txt; paste it in “Enhanced\_Sawtooth/pyclient”
6. Modify the value of “numBlocks” in the script.py to 10.
7. Terminal two (client): Submit the transactions by executing the code

```
$ python3 script.py
```

To generate the data points for the plots presented in the paper, batches with 1000 transactions, each having different dependencies among them, are submitted 20 times for each batch size. The script is then executed to measure the execution time for each run of the same batch.

## 4.2 Voting Transaction Family

Voting Transaction family has functionalities which can help to implement a voting system in blockchain. The key functionalities provided by this are:

We have stored sample test cases for voting transaction family in the folder “Enhanced\_Sawtooth/voting\_files”. To run please follow the below instructions:

#### Terminal one:

```
$ sudo docker-compose up
```

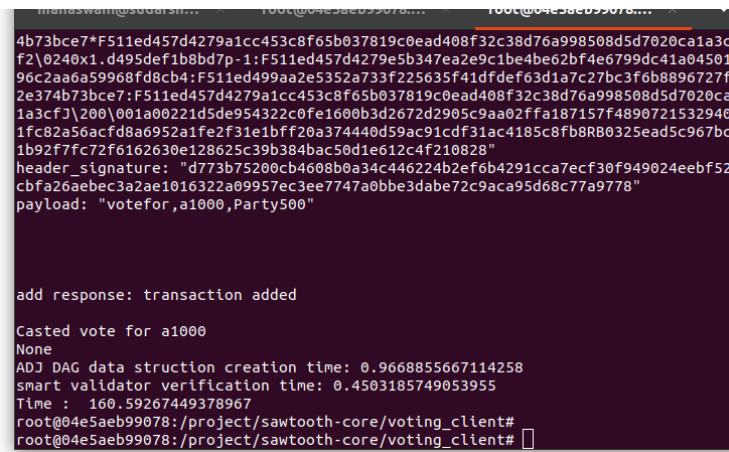
Wait for the node to create genesis block before starting the transaction family processor and clients.

**Terminal two (Client):**

```
$ sudo docker exec -it sawtooth-shell-local bash
$ cd voting_client
```

**Terminal three (Processor):**

```
$ sudo docker exec -it sawtooth-shell-local bash
$ cd voting_tp
$ python3 tp.py
```



```
4b73bce7*F511ed457d4279a1cc453c8f65b037819c0ead408f32c38d76a998508d5d7020ca1a3c
f2\0240x1.d495def1b8bd7p-1:F511ed457d4279e5b347ea2e9c1be4be62bf4e6799dc41a04501
96c2aa6a59968fd8cb4:F511ed499aa2e5352a733f225635f41dfdef63d1a7c27bc3f6b8896727f
2e374b73bce7:F511ed457d4279a1cc453c8f65b037819c0ead408f32c38d76a998508d5d7020ca
1a3cfj\200\001a00221d5de954322c0fe1600b3d2672d2905c9aa02ffa187157f4890721532940
1fc82a56acfd8a6952a1fe2f31e1bff20a374440d59ac91cdf31ac4185c8fb880325ead5c967bc
1b92f7fc72f6162630e128625c39b384bac50die612c4f210828"
header_signature: "d773b75200cb4608b0a34c446224b2ef6b4291cca7ecf30f949024eebf52
cbfa26aebec3a2ae1016322a09957ec3ee7747a0bbe3dabe72c9aca95d68c77a9778"
payload: "vote for, a1000, Party500"

add response: transaction added

Casted vote for a1000
None
ADJ DAG data struction creation time: 0.9668855667114258
smart validator verification time: 0.4503185749053955
Time : 160.59267449378967
root@04e5aeb99078:/project/sawtooth-core/voting_client#
root@04e5aeb99078:/project/sawtooth-core/voting_client#
```

Figure 10: Voting client terminal with total execution time and data structure creation time

#### automated script explanation:

1. The code removes the timing information from the file named "timing\_computation.txt". As the time for initial deposit batch execution is not required for the experiment.
2. The code contains a hard-coded value of 10 for the number of blocks. This means that the subsequent steps will be executed for a fixed number of blocks, which is 10 in this case. If you want to change the number of blocks, you can modify the value assigned to the variable "numBlocks" in the code.
3. The code proceeds to execute each block one by one. For each block, it performs batch submission and records the execution time in the "timing\_computation.txt" file.
4. Once all the executions for the 10 blocks are completed, the code extracts the execution times recorded in the "timing\_computation.txt" file. It then calculates the sum of the execution times for the 20 blocks. Since the blocks are executed twice the total time is for 20 executions.
5. Total time for the data structure creation for the smart validator plots.

#### Experiment one:

1. Go to the location “Enhanced\_Sawtooth/Complete\_Experiments/voting“
2. select one experiment and one conflict percentage.
3. Copy the content of folder and paste in voting\_client/batches.txt
4. Modify the value of “numBlocks“ in the script.py to 5.
5. Terminal two (client):Submit the transactions by executing the code

```
$ python3 script.py
```

To generate the data points for the plots displayed in the paper, the same batch is submitted multiple times, ranging from 10 to 50 times. The script is executed to obtain the execution time for each run of the same batch. In this particular case, running the script provided the execution time for 10 runs of the same batch.

#### Experiment two:

1. Go to the location “Enhanced\_Sawtooth/Complete\_Experiments/voting“
2. select one experiment and one conflict percentage.
3. Test files (200 to 1200) are provided.
4. Copy the contents of the folder and paste in voting\_client/batches.txt
5. Modify the value of “numBlocks“ in the script.py to 10.
6. Terminal two (client):Submit the transactions by executing the code

```
$ python3 script.py
```

To generate the data points for the plots presented in the paper, batches with transaction sizes ranging from 200 transactions to 1200 transactions are submitted a total of 20 times for each batch size. The script is executed to measure the execution time for each run of the same batch. In this specific scenario, running the script provided the execution time for 20 runs of each batch.

#### Experiment three:

1. Go to the location “Enhanced\_Sawtooth/Complete\_Experiments/voting“
2. select one experiment and one conflict percentage.
3. Test files (dep0 to dep10) are provided in location .
4. Copy the contents of the folder and paste in voting\_client/batches.txt
5. Modify the value of “numBlocks“ in the script.py to 10.
6. Terminal two (client):Submit the transactions by executing the code

```
$ python3 script.py
```

To generate the data points for the plots presented in the paper, batches with 1000 transactions, each having different dependencies among them, are submitted 20 times for each batch size. The script is then executed to measure the execution time for each run of the same batch.

## 5 Guide to the regenerating plots in the published work

To test the framework across different scenarios we have devised three conflict parameters (CP) that indicate the level of dependency among the transaction. Conflict parameter one (CP1) is the portion of transactions that have at least one dependency. CP2 ratio of dependencies (edges) to total possible dependencies. CP3 degree of parallelism in the DAG i.e., the number of independent components in the graph. We have designed four experiments, each varying one parameter while the rest of the parameters are constant. The four parameters are (1) number of blocks, (2) number of transaction in the block, (3) degree

of dependency and (4) number of threads. The experiment setup is named one to four respectively. We have named adjacency matrix implementation of our proposed framework as *Adj\_DAG*, linked list implementation as *LL\_DAG*. The Sawtooth inbuilt parallel scheduler uses tree data structure; accordingly we have named it as *Tree* and serial execution output as *Serial* in our results. For the paper we conducted each experiment five times and have used the average of the five values.

For Mixed transaction family we have created a new Sawtooth node with the schedulers and modified the block creation process to wait for three blocks of different transaction families before creating a block. Since this is not compatible with rest of the experiments, we have not shared it for artifact evaluation.

## 5.1 Experiment one guide

Follow the below steps to recreate Figure: 6, plot (a) of our report:

1. Look up the transaction family, experiment number and conflict parameter used for the experiment. In this case the details are Simplewallet, Experiment one and CP1.
2. The respective batch files are present in location "Enhanced\_Sawtooth/Complete\_Experiments". For this plot the inputs for the experiment are present inside "./simplewallet/Exp1/CP1".
3. Copy paste the folder contents into respective client of the transaction family. For simplewallet it is "Enhanced\_Sawtooth/pyclient folder" and for voting it is "Enhanced\_Sawtooth/voting\_client".
4. Set the "numBlocks" variable in the script to 5
5. Run the script files using the command :  

```
$ python3 script.py
```
6. The execution time for 10 blocks for the respective scheduler is displayed at the end of execution.
7. We have used this google sheets to create the plots. copy and paste the values in the relevant row and column. We have shared the template file named "Enhanced\_Sawtooth/plots", we have used with each sheet for each experiment.
8. Each run of this script gives execution time for 10 blocks, for execution time of 50, we had run the script 5 time and calculated the sum of all the five executions. (likewise for 20,30, and 40)

## 5.2 Experiment two guide

Follow the below steps to recreate Figure: 6, plot (b) of our report:

1. Look up the transaction family, experiment number and conflict parameter used for the experiment. In this case the details are Simplewallet, Experiment two and CP2.
2. The respective batch files are present in location "Enhanced\_Sawtooth/Complete\_Experiments". For this plot the inputs for the experiment are present inside "./simplewallet/Exp2/CP2".
3. For experiment two, there are multiple batches from 200 to 1200 transaction in the respective folders

4. Copy paste the folder contents into respective client of the transaction family. For simplewallet it is “Enhanced\_Sawtooth/pyclient folder” and for voting it is “Enhanced\_Sawtooth/voting\_client”.
5. Set the “numBlocks” variable in the script to 10
6. Run the script files using the command :
 

```
$ python3 script.py
```
7. The execution time for 20 blocks for the respective scheduler is displayed at the end of execution.
8. We have used this google sheets to create the plots. copy and paste the values in the relevant row and column. We have shared the template file named “Enhanced\_Sawtooth/plots”, we have used with each sheet for each experiment.
9. Each run of this script for each scheduler for each number of transactions 200, 400, 600, 800, 1000, and 1200.

### 5.3 Experiment three guide

Follow the below steps to recreate Figure: 6, plot (c) of our report:

1. Look up the transaction family, experiment number and conflict parameter used for the experiment. In this case the details are Simplewallet, Experiment three and CP3.
2. The respective batch files are present in location “Enhanced\_Sawtooth/Complete\_Experiments”. For this plot the inputs for the experiment are present inside “./simplewallet/Exp3/CP3”.
3. For experiment three, there are multiple batches with dep 0 to dep 10 in the respective folders
4. Copy paste the folder contents into respective client of the transaction family. For simplewallet it is “Enhanced\_Sawtooth/pyclient folder” and for voting it is “Enhanced\_Sawtooth/voting\_client”.
5. Set the “numBlocks” variable in the script to 10
6. Run the script files using the command :
 

```
$ python3 script.py
```
7. The execution time for 20 blocks for the respective scheduler is displayed at the end of execution.
8. We have used this google sheets to create the plots. copy and paste the values in the relevant row and column. We have shared the template file named “Enhanced\_Sawtooth/plots”, we have used with each sheet for each experiment.
9. Each run of this script for each scheduler for each number of transactions dep 0, dep 2, dep 4, dep 6, dep 8, and dep 10.
10. For experiment three we have used throughput instead of execution time as Y-axis. for this one need to divide number of transaction by total time, here its 20000/execution time (20 blocks, 1000 transactions per block so 20000 transactions).
11. The modified experiments 3 sheet in the file “Enhanced\_Sawtooth/plots” does this using formulas



## 5.4 Data structure creation

The plots from Figure 6 (i),(j), and (k) are from simplewallet transaction family CP1 experiments 1,2, and 3. When you run the scripts for the respective experiments the data creation time is also displayed in the output Figure 10. Use the file “Enhanced\_Sawtooth/plots\_DS” to recreate the plots from the paper. The smart validator data creation time is displayed when using the matrix adjacency scheduler. Copy paste the values into respective positions in the sheets. The third experiment also depicts execution time as Y-axis so the modified experiments sheet is not required in this case. Serial scheduler is not required as this scheduler doesn’t create a data structure for preprocessing.

## 6 Interpreting the logs without using scripts

If the user has submitted a batch without using scripts. The log files can be utilized to understand the execution time and dataset creation time of the submitted batch. The information in Figure: 11 is how the log is created. We have explained each of them line by line:

Line 1 - Block production: 1.018228769302368164

This refers to the time it takes to execute the genesis block, which is not relevant to our experiment. We have shown it for completeness.

**Execution time for batch submitted:**

Line 2 - Block production: 11.707515954971313

Line 3 - Block validation: 11.432829141616821

Lines 2 and 3 in the provided context refer to the time taken for block production and block validation for a batch that has been submitted. These timings are computed by the system and are typically appended to log files.

```
Computations_timing.txt
-----
1. execution:1.018228769302368164
-----
2. execution:11.707515954971313
-----
3. execution:11.432829141616821
-----
```

Figure 11: timing\_computation.txt file

To ensure that only the relevant data is present in the log files, it is advised to delete the log files before submitting a new block. By doing so, you can avoid any interference or confusion caused by previous block production and validation timings that might be present in the log files.

Deleting the log files before submitting a new block allows you to focus on the accurate and up-to-date timings for the current batch being processed. This practice ensures that you have clear and relevant information available for analysis or any further actions related to block production and validation.