

Performance Testing for Liquidity Module

B-Harvest

1. Introduction

- Core Tendermint & Cosmos-SDK computations for Liquidity Module
 - Tendermint :
 - Consensus process among validators
 - P2P communications for transaction relaying
 - Cosmos-SDK Ante :
 - Signature verification
 - Gas Calculation
 - Cosmos-SDK Bank :
 - Multisend
 - Supply
 - Liquidity Module :
 - Swap message storage
 - Batch process computation
- Main Objectives
 - To test the computation performance of Liquidity Module
 - Figure out bottleneck points if there exists significant performance problems
 - Addressing solutions to solve performance problems by computation algorithm optimization in Liquidity Module
- No Goal
 - Performance delay due to bottlenecks outside Liquidity Module is out of scope for investigation

2. Testing Setup

- Hardwares
 - Instance : AWS m5.Large (2 Core, 8Gb Ram)
 - Location : US West, US East, Europe, Asia
 - Sentry Architecture : 1 validator and 1 sentry in each location
- Liquidity Module
 - [Milestone 2 WIP version](#)
 - cosmos-sdk v0.40.0
 - tendermint v0.34.1
- Swap Bot Model
 - Random Order Price
 - Every order has different order prices
 - Statistically, half of submitted orders are executed
 - Number of Accounts
 - We utilized 6 accounts to submit swap orders
 - Bulk / Splitted Transactions : Two different simulations
 - Bulk Transactions
 - Large number of messages in one transaction(hence small number of transactions)
 - Purely testing Liquidity Module performance(Minimal burden on transaction handling)
 - Splitted Transactions
 - Low number of messages in one transaction(hence large number of transactions)
 - Performance might be impacted by the basic transaction handling processes in Tendermint and Cosmos-SDK
- <https://github.com/b-harvest/swapbot>

3. Results

- Below result is a simulation result statistics with given scenarios
- We assumed batch period as one block(batch processed every block)
- Swap orders submitted to one liquidity pool to stress-test performance when swap orders are concentrated into one liquidity pool

Simulation Type	# of msgs (per block)	# of txs (per block)	Average Block Time	Average CPU Usage	tx Missing (per block)
No tx	0	0	6.19s	1%	-
Bulk 1	102	6	6.27s	2%	-
Splitted 1	102	102	6.46s	5.5%	-
Bulk 2	198	6	6.38s	4.5%	-
Splitted 2	198	198	6.66s	15%	-
Bulk 3	396	6	6.60s	12%	-
Splitted 3	396	396	6.90s	20%	-
Bulk 4	798	6	6.88s	25%	-
Splitted 4	798	798	7.41s	42%	148
Bulk 5	1,596	6	7.21s	50%	-
Splitted 5	1,596	1,596	7.88s	75%	101
Bulk 6	3,198	6	7.90s	85%	-
Splitted 6	3,198	3,198	7.83s	120%	1,146
Bulk 6	6,396	6	9.05s	120%	1
Splitted 6	6,396	6,396	-	-	too many
Bulk(Send)	798	6	6.62s	8%	-
Splitted(Send)	798	798	7.16s	40%	-

4. Conclusion

- We observed that in Splitted Transaction scenarios, validators are **starting to miss significant amount of transactions** to be included in each block when number of transactions in a block increased to **more than 700 txs**
- But, in Bulk Transaction scenarios, where **the impact of Liquidity Module computation is more purely affected**, transaction missing is not happening even in 3,198 messages per block scenarios
- We have block time delay with **fairly less than linear correlation** with number of messages in a block (Bulk Transaction scenarios)
 - **1.01** second delay in **1,596** msgs per block
 - **1.71** second delay in **3,198** msgs per block
 - **2.86** second delay in **6,396** msgs per block
- In 798 msgs per block scenarios, if we compare swap messages and send messages, the block time difference for Bulk/Splitted simulations are **0.26s** and **0.25s** respectively. We conclude that **these differences are not significant** considering much more computation processes existing for swap messages than send messages