Performance analysis project on Hyperledger Fabric Network

with Term Paper on

Performance Evaluation of Permissioned Blockchain Platforms

Submitted by:
Ankit RAJ
SAU/M(CS)/2020/04

Submitted to:
Dr. Shachi Sharma
Department of Computer Science
South Asian University, New Delhi, India

Performance Evaluation of Permissioned Blockchain Platforms

Abstract

Blockchain at its basic level distributed network of databases that store transactions that replicate and spread over the entire network of computer systems on the blockchain. This technology eliminates the need for centralized database. Every block in the blockchain contains a number of transactions, and every time a new transaction occurs on the blockchain, a record of that transaction is added to every participant's ledger. The decentralised database managed by multiple participants is known as Distributed Ledger Technology (DLT). Blockchain technology eliminates the limitation of traditional ledger system, such as single point of failure and dependence on third party and also improves the fault tolerance. Blockchain technology over time have evolved into public, private blockchain. Private blockchain provides additional security and possibility of using this new technology in variety of application. There are many permissioned blockchain platforms today like Ethereum, Quorum, Parity, MultiChain, Hyperledger Fabric etc.

I am going to investigate the performance evaluation of Hyperledger fabric, since performance evaluation for a blockchain network requires very extensive analysis, which is very daunting and require support of powerful machines, so I am going to first summarise the results and findings of various research article. In the later part I have also conducted a very rudimentary performance analysis over my simple fabric network to illustrate the process of analysis.

I. Introduction

Blockchain technology was first introduced to this world as the underlying technology behind the Bitcoin – worlds first decentralised cryptocurrency. By its distributed nature, blockchain overcomes limitations of centralised system; each entity can have a copy of the ledger, and all the entities should agree on the transactions before they are committed to the chain. In the blockchain, each block of transactions refers to the previous block by its hash, which guarantees data integrity. Thus, if an attacker tries to modify any block, the change will propagate through the chain and will be recognized. Moreover, blockchain enables the usage of smart contracts. A smart contract or a digital contract is an automated program that controls the transaction logic and does not require an intermediary to run it Now a days Blockchain is not limited to cryptocurrencies but industries have adopted its potential to be used in different areas like banking, healthcare, supply chains etc. There are various blockchain platform which allows user to develop blockchain application for their use. In this paper we will concentrate on Hyperledger Fabric.

Before moving to next section lets get familiar with some terms related to blockchain

Audience: smart contract developers, architects, application and administrators

Digital Assets: There is a user-incentivize tokens that can increased or decreased in value based on the relevance and state of the blockchain they belong to, in a permissionless network, . These blockchains can either employ monetary or utility tokens, depending on the purpose.

Block: Transactions are batched together and combined into single blocks at every fixed interval of time, this new block can have a fixed defined size . Every block in a blockchain contains: a reference to the previous block, a timestamp, a summary of the included transaction and the Proof of Work (see ‘6’ below) that went into creating the secure block.

Orderer: The Orderer is a type of peer of node who is responsible for packaging transactions into Blocks, and distribute them to Anchor Peers across the network

Policy: policy is a set of rules that define the structure for how decisions are made and specific outcomes are reached. It describes **who** and **what**, what access or rights an individual has over the **assets**. We can see these policies are used throughout our daily lives to protect assets of value to us, from car rentals, health, our homes, and many more.

Peer/Nodes: A blockchain network is primarily comprised of a set of *peer nodes*. Peers are a fundamental elements of the network because they host smart contracts and ledgers . Ledgers immutably records all the transactions generated by smart contracts. Smart contracts and ledgers are used to encapsulate the shared *processes* and shared *information* in a network, respectively. These aspects of a peer make them a good starting point to understand a Fabric network.

Ledger: A ledger is the journel of transactions that contains the current state of a business.
Proof of Work: Proof of work (PoW) is a form of cryptographic zero-knowledge proof in which one party (the prove) proves to others (the verifiers) that a certain amount of a specific computational effort has been expended. Verifiers subsequently confirms this expenditure with minimal effort on their part. Many cryptocurrencies, including Bitcoin and .Ethereum is secured by POW algorithm. while most of digital currencies have a central entity or leader keeping track of every user and how much money they have. But there's no such leader in charge of cryptocurrencies like Bitcoin. ofr such cases Proof-of-work is needed to make the online currency work without a company or government running the show.

Smart Contract / Chaincode: *Smart contract is executable rules between different organizations. Smart contract gets invoked by application to generate transactions that are recorded on the ledger*. Since a smart contract can implement the governance rules for **any** type of business regulation, so they are automatically enforced when the smart contract is executed.
In terms of Hyperladger fabric this Smart contract is represented by Chaincode.

A. HYPERLEDGER FABRIC

Fabric is extensible open-source system for deploying and operating permissioned blockchains and one of the Hyperledger projects hosted by the Linux Foundation. Fabric adopts a new kind of transaction architecture, Execute Order-Validate architecture, which distinguish it from other blockchain platforms. It has modular consensus protocols, which allows the system to be tailored for particular use cases and trust models. It is also the first blockchain platform which runs distributed applications written in general-purpose programming languages, without systemic dependency on a native cryptocurrency. This stands in sharp contrast to existing blockchain platforms that require “smart-contracts” to be written in domain specific languages or rely on a cryptocurrency. Fabric realizes the permissioned model using a portable notion of membership, which can be integrated with industry standard identity management. To support such flexibility, Fabric introduces an entirely novel blockchain design and revamps the way blockchains cope with non-determinism, resource exhaustion, and performance attacks. To eliminate non-deterministic operation Hperledger fabric allows smart contracts or chaincode as in fabric terms to be written in domain specific language(DSL)

Key Components of Fabric

A. Peer: Nodes who executes and implements chaincode or smart contract, manages ledger on their file system

B. Channels: A channel is a private blockchain overlay which allows for data isolation and confidentiality. A channel-specific ledger is shared across the peers in the channel, and transacting parties must be authenticated to a channel in order to interact with it.

C. Ordering services: For consensus purpose HLF have ordering nodes, their main responsibility is ordering the transactions in the blocks and broadcasting them them to the peers for validations.

HLF have three type of ordering services these are:

a.) Solo
b.) Kafka-zookeeper
c.) Raft

E. Transaction Flow in HLF(Hyperledger Fabric)

In endorsing phase each transaction goes through the phase if Execute-Order-Validate phase. This process starts from transaction proposal and ends with its commit to ledger

* Endorsement Phase or Proposal: in this phase application sends transaction request to endorsing peers, these peer check if: proposal is well structures, follows the convention, not trying to duplicate any transaction, signature is valid and is the request comes from valid peer of not.

After the peers have checked the validity signed transaction is send back to the application this ends the endorsement phase of Transaction.

* Ordering Phase: In this phase orderers package the transaction into the blocks, no of transaction per block is defined by channel configuration and it affects the overall latency of admitting the transaction into the blockchain.

* Validation Phase: In this phase orderer sends the block to the peers, the peers who are offline will receive the block by gossiping. Each peer will validate the block in same way this makes ledger present to all identical. To commit block to ledgers peers performs consistency checks. Each valid block will be decoded and all transactions in the blocks will be validated by VSCC validation then MVCC validation will be performed.

Ledger Update phase : In this phase local ledgers will be updated, these update will be carried atomically on StateDB where all keys and write sets of valid transactions are stored.

F. Configuration Parameters

We will study different configuration parameter that can affect the HLF network performance. Some of important configuration that should be considered are:

* Block Size: Transactions are batched together in block and each block reference the hash of its previous block. For a blockchain system block size plays a major role in their performance.

* Endorsement Policy: Endorsement policy decides the no of transactions that should be dome before we commit to the block and some crucial rules in the transactions. The complexity of endorsement policy affects resources and the time taken to collect and evaluate a blockchain network.

* Channels: channels allows the inner abstraction and additional chaincode execution, by this way channels brings isolation among transactions and parallelism in network as whole. No of channels also have a significant effect on the platform performance and scalability.

* Resource Allocation: peer runs the complex CPU intensive signature computation and verification routines, varying resource of peer system have also a major effect in blockchain platform performance

* Ledger Database: There are two ledger key-value storage database- CouchDB and GoLevelDB, their performance also affects the overall performance

II. PROBLEM STATEMENT

The Fabric comprises of various components such as smart contracts, endorsers, committers, validators, and orderers to name few. As the performance of blockchain platform is a major concern for enterprise applications, in this work, I will summarise some empirical study to characterize the performance of Hyperledger Fabric and identify potential performance bottlenecks to gain a better understanding of the system. Also a mini analysis on Hyperledger fabric simple network would be carried.

To conduct an in-depth study of Fabric core components and benchmark Fabric performance for common usage patterns. We aim to study the throughput and latency characteristics of the system when varying the configuration of parameters to understand the relationship between the performance metrics and parameters.

* Performance matrices

a.) Throughput: Rate at which transactions are committed to ledgers.
b.) Latency: Time take from transaction proposal to the completion of the transaction. following latencies contribute in the overall latency:
* Endorsement Latency
* Broadcast latency
* Commit Latency
* Ordering Latency
* VSCC Validation Latency
* MVCC Validation Latency
* Ledger Update Latency

IV. EXPERIMENT SETUP

A. Setup and Workloads

Experimental setup of my Project on the performance benchmarking of HLF Blockchain network

I have used three factors : HLF Network System version, No of transactions and no of records per block for studying the performance of fabric networks, with some fixed configuration

* Single host,
* 2 organisation with one peer.
* Endorsement policy: Raft
* StateDB: CouchDB

Performance Matrices Considered

1. Throughput
2. Latency

Factors and their levels

Factors =>	Blockchain Network Version (A)	No of transactions (B)	Transactions per block (C)
Level (1)	HLF 1.4	300	1
Level (-1)	HLF 2.0	400	2

Since we have 2 factors with 2 levels each so I will conduct 2^3= 8 experiments and analyse the experimental results using 2k factorial design.

In this system i have evaluated the Fabcar HLF network this network runs over docker container and containers act as the node in network.

Hyperledger Caliper benchmark is used to generate the report of performance over the same workload for HLF 1.40 and HLF 2.0 Fabcar Network:

workload Script to generate our report is provided below:

```
'use strict';

module.exports.info = 'opening accounts';
const { v1: uuidv4 } = require('uuid')

let account_array = [];

let bc, contx;
var txnPerBatch = 1
module.exports.init = function (blockchain, context, args) {
  if (!args.hasOwnProperty('txnPerBatch')) {
    args.txnPerBatch = 1;
  }
  txnPerBatch = args.txnPerBatch;
  bc = blockchain;
  contx = context;

  return Promise.resolve();
};


function generateWorkload() {
  let workload = [];
  for (let i = 0; i < txnPerBatch; i++) {
    let id = uuidv4();
    account_array.push(id)

    workload.push({
      chaincodeFunction: 'createCar',
      chaincodeArguments: [id, 'A', 'B', 'C', "d"],
    });
  }
  return workload;
}

module.exports.run = function () {
  let args = generateWorkload();
  return bc.invokeSmartContract(contx, 'fabcar', '1', args);
};

module.exports.end = function () {
  return Promise.resolve();
};
```

```
module.exports.account_array = account_array;
```

```
'use strict';

module.exports.info  = 'querying accounts';


let bc, contx;
let account_array;

module.exports.init = function(blockchain, context, args) {
  const open = require('./open.js');
  bc        = blockchain;
  contx     = context;
  account_array = open.account_array;

  return Promise.resolve();
};

module.exports.run = function() {
  const acc   = account_array[Math.floor(Math.random()*(account
_array.length))];

    let args = {
      chaincodeFunction: 'queryCar',
      chaincodeArguments: [acc],
    };

    return bc.bcObj.querySmartContract(contx, 'fabcar', '1', a
rgs, 10);
};

module.exports.end = function() {
  // do nothing
  return Promise.resolve();
};
```

Workloads Script for our HLF 2.0

```
'use strict';

const { WorkloadModuleBase } = require('@hyperledger/caliper-core');

const colors = ['blue', 'red', 'green', 'yellow', 'black', 'purple', 'white', 'violet', 'indigo', 'brown
'];
const makes = ['Toyota', 'Ford', 'Hyundai', 'Volkswagen', 'Tesla', 'Peugeot', 'Chery', 'Fiat', 'Tata', '
Holden'];
const models = ['Prius', 'Mustang', 'Tucson', 'Passat', 'S', '205', 'S22L', 'Punto', 'Nano', 'Barina'];
const owners = ['Tomoko', 'Brad', 'Jin Soo', 'Max', 'Adrianna', 'Michel', 'Aarav', 'Pari', 'Valeria', 'S
hotaro'];

/**
 * Workload module for the benchmark round.
 */
class CreateCarWorkload extends WorkloadModuleBase {
    /**
     * Initializes the workload module instance.
     */
    constructor() {
        super();
        this.txIndex = 0;
    }

    /**
     * Assemble TXs for the round.
     * @return {Promise<TxStatus[]>}
     */
    async submitTransaction() {
        this.txIndex++;
        let carNumber = 'Client' + this.workerIndex + '_CAR' + this.txIndex.toString();
        let carColor = colors[Math.floor(Math.random() * colors.length)];
        let carMake = makes[Math.floor(Math.random() * makes.length)];
        let carModel = models[Math.floor(Math.random() * models.length)];
        let carOwner = owners[Math.floor(Math.random() * owners.length)];

        let args = {
            contractId: 'fabcar',
            contractVersion: 'v1',
            contractFunction: 'createCar',
            contractArguments: [carNumber, carMake, carModel, carColor, carOwner],
            timeout: 30
        };

        await this.sutAdapter.sendRequests(args);
    }
}

/**
 * Create a new instance of the workload module.
 * @return {WorkloadModuleInterface}
 */
```

```
function createWorkloadModule() {
   return new CreateCarWorkload();
}

module.exports.createWorkloadModule = createWorkloadModule;
```

```
'use strict';

module.exports.info = 'opening accounts';
const { v1: uuidv4 } = require('uuid')

let account_array = [];

let bc, contx;
var txnPerBatch = 1
module.exports.init = function (blockchain, context, args) {
   if (!args.hasOwnProperty('txnPerBatch')) {
      args.txnPerBatch = 1;
   }
   txnPerBatch = args.txnPerBatch;
   bc = blockchain;
   contx = context;

   return Promise.resolve();
};


function generateWorkload() {
   let workload = [];
   for (let i = 0; i < txnPerBatch; i++) {

      workload.push({
         chaincodeFunction: 'createCar',
         chaincodeArguments: [uuidv4(), 'A', 'B', 'C', "d"],
      });
   }
   return workload;
}

module.exports.run = function () {
   let args = generateWorkload();
   return bc.invokeSmartContract(contx, 'fabcar', '1', args);
};

module.exports.end = function () {
   return Promise.resolve();
};

module.exports.account_array = account_array;
```

```
'use strict';

const { WorkloadModuleBase } = require('@hyperledger/caliper-core');

// const helper = require('./helper');

/**
 * Workload module for the benchmark round.
 */
class QueryCarWorkload extends WorkloadModuleBase {
   /**
    * Initializes the workload module instance.
    */
   constructor() {
      super();
      this.txIndex = 0;
      this.limitIndex = 0;
   }

   /**
    * Initialize the workload module with the given parameters.
    * @param {number} workerIndex The 0-based index of the worker instantiating the workload module.
    * @param {number} totalWorkers The total number of workers participating in the round.
    * @param {number} roundIndex The 0-based index of the currently executing round.
    * @param {Object} roundArguments The user-provided arguments for the round from the benchmark confi
guration file.
    * @param {BlockchainInterface} sutAdapter The adapter of the underlying SUT.
    * @param {Object} sutContext The custom context object provided by the SUT adapter.
    * @async
    */
   // async initializeWorkloadModule(workerIndex, totalWorkers, roundIndex, roundArguments, sutAdapter,
sutContext) {
   //      await super.initializeWorkloadModule(workerIndex, totalWorkers, roundIndex, roundArguments,
sutAdapter, sutContext);

   //      this.limitIndex = this.roundArguments.assets;
   //      await helper.createCar(this.sutAdapter, this.workerIndex, this.roundArguments);
   // }

   /**
    * Assemble TXs for the round.
    * @return {Promise<TxStatus[]>}
    */
   async submitTransaction() {
      this.txIndex++;
      let carNumber = 'Client' + this.workerIndex + '_CAR' + this.txIndex.toString();

      let args = {
         contractId: 'fabcar',
         contractVersion: 'v1',
         contractFunction: 'queryCar',
         contractArguments: [carNumber],
         timeout: 30,
```

```
        readOnly: true
      };

      if (this.txIndex === this.limitIndex) {
        this.txIndex = 0;
      }

      await this.sutAdapter.sendRequests(args);
    }
}

/**
 * Create a new instance of the workload module.
 * @return {WorkloadModuleInterface}
 */
function createWorkloadModule() {
    return new QueryCarWorkload();
}

module.exports.createWorkloadModule = createWorkloadModule;
```

These workload script runs over caliper benchmark and caliper generates the detailed report of our resource utilisation and performance of corresponding experiment

Resource utilisation and performance matric captured using Hyperledger caliper in different setup is shown below

Experiment 1 System HLF 1.4, Total transactions 300, No of transactions perblock 1

Summary of performance metrics

Name	Succ	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
open	300	0	39.8	8.30	0.86	6.21	24.4

Resource monitor: docker

Name	CPU% (max)	CPU% (avg)	Memory(max) [MB]	Memory(avg) [MB]	Traffic In [MB]	Traffic Out [MB]	Disc Write [MB]	Disc Read [MB]
dev-peer0.org2.example.com-fabcar-1	3.77	1.20	11.8	9.91	0.642	0.379	0.00	1.66
dev-peer0.org1.example.com-fabcar-1	3.51	1.15	12.2	9.63	0.609	0.336	0.00	1.74
caliper	0.00	0.00	0.00	0.00	0.00	-	0.00	0.00
peer0.org1.example.com	31.9	15.31	112	152	2.87	3.42	1.90	2.04
ca.org2.example.com	0.00	0.00	5.79	5.65	0.000175	0.00	0.00	0.00
orderer.example.com	5.17	1.97	34.5	32.7	2.31	2.34	3.64	0.922
couchdb2	37.21	10.92	32.3	40.8	0.176	0.194	1.26	0.0664
couchdb1	37.68	15.67	44.1	45.9	0.239	0.266	1.57	0.0977
couchdb0	37.79	13.15	49.9	48.5	0.192	0.213	1.30	0.0664
ca.org1.example.com	0.00	0.00	16.5	16.3	0.000175	0.00	0.00	0.00
peer1.org2.example.com	10.62	5.56	66.8	59.7	1.96	0.394	2.55	2.10
orderer2.example.com	10.99	4.02	39.1	30.8	1.86	3.44	3.65	7.40
peer0.org2.example.com	34.76	15.64	90.9	85.3	2.89	3.19	1.72	0.582
orderer3.example.com	8.87	3.29	34.8	29.4	2.29	2.32	3.65	0.719
cli	0.00	0.00	2.95	2.94	0.000212	0.00	0.00	0.00
peer1.org1.example.com	22.83	10.15	81.8	76.4	1.94	0.333	2.36	0.102
couchdb3	37.93	16.26	46.7	45.2	0.259	0.295	1.70	0.109

Experiment 2 System HLF 2.0, Total transactions 300, No of transactions perblock 1

Summary of performance metrics

Name	Succ	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
open	300	0	40.1	6.04	0.74	5.53	26.7

Resource monitor: docker

Name	CPU% (max)	CPU% (avg)	Memory(max) [MB]	Memory(avg) [MB]	Traffic In [MB]	Traffic Out [MB]	Disc Write [MB]	Disc Read [MB]
dev-peer0.org2.example.com-fabcar-1	3.49	1.13	12.5	9.47	0.712	0.347	0.00	1.86
dev-peer0.org1.example.com-fabcar-1	3.30	1.05	12.0	9.43	0.709	0.336	0.00	1.84
caliper	0.00	0.00	0.00	0.00	0.00	-	0.00	0.00
peer0.org1.example.com	32.28	15.27	105	102	2.87	3.42	1.90	2.04
ca.org2.example.com	0.00	0.00	5.79	5.65	0.000212	0.00	0.00	0.00
orderer.example.com	5.21	1.96	35.5	30.7	2.31	2.34	3.64	0.922
couchdb2	37.21	10.92	52.3	50.8	0.176	0.194	1.26	0.0664
couchdb1	37.68	15.37	54.1	52.7	0.239	0.266	1.57	0.0977
couchdb0	37.79	12.95	49.2	48.5	0.192	0.213	1.30	0.0664
ca.org1.example.com	0.00	0.00	15.5	15.3	0.000212	0.00	0.00	0.00
peer1.org2.example.com	10.62	5.56	66.8	59.7	1.96	0.394	2.55	2.10
orderer2.example.com	10.79	3.98	38.5	30.8	1.86	3.44	3.65	7.40
peer0.org2.example.com	34.76	15.84	90.9	85.3	2.89	3.19	1.72	0.582
orderer3.example.com	8.97	3.29	34.8	29.4	2.29	2.32	3.65	0.719
cli	0.00	0.00	2.85	2.84	0.000212	0.00	0.00	0.00
peer1.org1.example.com	22.83	10.15	81.8	76.4	1.94	0.333	2.36	0.102
couchdb3	37.93	16.26	46.7	45.2	0.259	0.295	1.70	0.109

Experiment 3 System: HLF 1.4, Total transactions 400, No of transactions perblock 1

Summary of performance metrics

Name	Succ	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
open	400	0	39.8	8.36	0.96	6.11	24.6

Resource monitor: docker

Name	CPU% (max)	CPU% (avg)	Memory(max) [MB]	Memory(avg) [MB]	Traffic In [MB]	Traffic Out [MB]	Disc Write [MB]	Disc Read [MB]
dev-peer0.org2.example.com-fabcar-1	3.67	1.19	12.8	9.90	0.612	0.367	0.00	1.66
dev-peer0.org1.example.com-fabcar-1	3.51	1.15	12.2	9.63	0.609	0.336	0.00	1.74
caliper	0.00	0.00	0.00	0.00	0.00	-	0.00	0.00
peer0.org1.example.com	32.74	15.31	112	112	2.87	3.42	1.90	2.04
ca.org2.example.com	0.00	0.00	5.79	5.65	0.000212	0.00	0.00	0.00
orderer.example.com	5.21	1.96	35.5	30.7	2.31	2.34	3.64	0.922
couchdb2	37.21	10.92	52.3	50.8	0.176	0.194	1.26	0.0664
couchdb1	37.68	15.67	54.1	52.9	0.239	0.266	1.57	0.0977
couchdb0	37.79	13.15	49.9	48.5	0.192	0.213	1.30	0.0664
ca.org1.example.com	0.00	0.00	16.5	16.3	0.000212	0.00	0.00	0.00
peer1.org2.example.com	10.62	5.56	66.8	59.7	1.96	0.394	2.55	2.10
orderer2.example.com	10.99	4.02	39.1	30.8	1.86	3.44	3.65	7.40
peer0.org2.example.com	34.76	15.64	90.9	85.3	2.89	3.19	1.72	0.582
orderer3.example.com	8.87	3.29	34.8	29.4	2.29	2.32	3.65	0.719
cli	0.00	0.00	2.95	2.94	0.000212	0.00	0.00	0.00
peer1.org1.example.com	22.83	10.15	81.8	76.4	1.94	0.333	2.36	0.102
couchdb3	37.93	16.26	46.7	45.2	0.259	0.295	1.70	0.109

Experiment 4 System: HLF 2.0, Total transactions 400, No of transactions perblock 1

Summary of performance metrics

Name	Succ	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
open	400	0	39.6	6.17	0.81	6.02	26.9

Resource monitor: docker

Name	CPU% (max)	CPU% (avg)	Memory(max) [MB]	Memory(avg) [MB]	Traffic In [MB]	Traffic Out [MB]	Disc Write [MB]	Disc Read [MB]
dev-peer0.org2.example.com-fabcar-1	3.49	1.13	12.5	9.47	0.712	0.347	0.00	1.86
dev-peer0.org1.example.com-fabcar-1	3.30	1.05	12.0	9.43	0.709	0.336	0.00	1.84
caliper	0.00	0.00	0.00	0.00	0.00	-	0.00	0.00
peer0.org1.example.com	32.28	15.27	105	102	2.87	3.42	1.90	2.04
ca.org2.example.com	0.00	0.00	5.79	5.65	0.000212	0.00	0.00	0.00
orderer.example.com	5.21	1.96	35.5	30.7	2.31	2.34	3.64	0.922
couchdb2	37.21	10.92	52.3	50.8	0.176	0.194	1.26	0.0664
couchdb1	37.68	15.37	54.1	52.7	0.239	0.266	1.57	0.0977
couchdb0	37.79	12.95	49.2	48.5	0.192	0.213	1.30	0.0664
ca.org1.example.com	0.00	0.00	15.5	15.3	0.000212	0.00	0.00	0.00
peer1.org2.example.com	10.62	5.56	66.8	59.7	1.96	0.394	2.55	2.10
orderer2.example.com	10.79	3.98	38.5	30.8	1.86	3.44	3.65	7.40
peer0.org2.example.com	34.76	15.84	90.9	85.3	2.89	3.19	1.72	0.582
orderer3.example.com	8.97	3.29	34.8	29.4	2.29	2.32	3.65	0.719
cli	0.00	0.00	2.85	2.84	0.000212	0.00	0.00	0.00
peer1.org1.example.com	22.83	10.15	81.8	76.4	1.94	0.333	2.36	0.102
couchdb3	37.93	16.26	46.7	45.2	0.259	0.295	1.70	0.109

Experiment 5 System: HLF 1.4, Total transactions 300, No of transactions perblock 2

Summary of performance metrics

Name	Succ	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
open	300	0	40.1	7.16	0.61	6.03	25.4

Resource monitor: docker

Name	CPU% (max)	CPU% (avg)	Memory(max) [MB]	Memory(avg) [MB]	Traffic In [MB]	Traffic Out [MB]	Disc Write [MB]	Disc Read [MB]
dev-peer0.org2.example.com-fabcar-1	3.67	1.19	12.8	9.90	0.612	0.367	0.00	1.66
dev-peer0.org1.example.com-fabcar-1	3.51	1.15	12.2	9.63	0.609	0.336	0.00	1.74
caliper	0.00	0.00	0.00	0.00	0.00	-	0.00	0.00
peer0.org1.example.com	32.74	15.31	112	112	2.87	3.42	1.90	2.04
ca.org2.example.com	0.00	0.00	5.79	5.65	0.000212	0.00	0.00	0.00
orderer.example.com	5.21	1.96	35.5	30.7	2.31	2.34	3.64	0.922
couchdb2	37.21	10.92	52.3	50.8	0.176	0.194	1.26	0.0664
couchdb1	37.68	15.67	54.1	52.9	0.239	0.266	1.57	0.0977
couchdb0	37.79	13.15	49.9	48.5	0.192	0.213	1.30	0.0664
ca.org1.example.com	0.00	0.00	16.5	16.3	0.000212	0.00	0.00	0.00
peer1.org2.example.com	10.62	5.56	66.8	59.7	1.96	0.394	2.55	2.10
orderer2.example.com	10.99	4.02	39.1	30.8	1.86	3.44	3.65	7.40
peer0.org2.example.com	34.76	15.64	90.9	85.3	2.89	3.19	1.72	0.582
orderer3.example.com	8.87	3.29	34.8	29.4	2.29	2.32	3.65	0.719
cli	0.00	0.00	2.95	2.94	0.000212	0.00	0.00	0.00
peer1.org1.example.com	22.83	10.15	81.8	76.4	1.94	0.333	2.36	0.102
couchdb3	37.93	16.26	46.7	45.2	0.259	0.295	1.70	0.109

Experiment 6 System: HLF 2.0, Total transactions 300, No of transactions perblock 2

Summary of performance metrics

Name	Succ	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
open	300	0	40.1	6.24	0.74	5.53	26.9

Resource monitor: docker

Name	CPU% (max)	CPU% (avg)	Memory(max) [MB]	Memory(avg) [MB]	Traffic In [MB]	Traffic Out [MB]	Disc Write [MB]	Disc Read [MB]
dev-peer0.org2.example.com-fabcar-1	3.49	1.13	12.5	9.47	0.712	0.347	0.00	1.86
dev-peer0.org1.example.com-fabcar-1	3.30	1.05	12.0	9.43	0.709	0.336	0.00	1.84
caliper	0.00	0.00	0.00	0.00	0.00	-	0.00	0.00
peer0.org1.example.com	32.28	15.27	105	102	2.87	3.42	1.90	2.04
ca.org2.example.com	0.00	0.00	5.79	5.65	0.000212	0.00	0.00	0.00
orderer.example.com	5.21	1.96	35.5	30.7	2.31	2.34	3.64	0.922
couchdb2	37.21	10.92	52.3	50.8	0.176	0.194	1.26	0.0664
couchdb1	37.68	15.37	54.1	52.7	0.239	0.266	1.57	0.0977
couchdb0	37.79	12.95	49.2	48.5	0.192	0.213	1.30	0.0664
ca.org1.example.com	0.00	0.00	15.5	15.3	0.000212	0.00	0.00	0.00
peer1.org2.example.com	10.62	5.56	66.8	59.7	1.96	0.394	2.55	2.10
orderer2.example.com	10.79	3.98	38.5	30.8	1.86	3.44	3.65	7.40
peer0.org2.example.com	34.76	15.84	90.9	85.3	2.89	3.19	1.72	0.582
orderer3.example.com	8.97	3.29	34.8	29.4	2.29	2.32	3.65	0.719
cli	0.00	0.00	2.85	2.84	0.000212	0.00	0.00	0.00
peer1.org1.example.com	22.83	10.15	81.8	76.4	1.94	0.333	2.36	0.102
couchdb3	37.93	16.26	46.7	45.2	0.259	0.295	1.70	0.109

Experiment 7 System: HLF 1.4, Total transactions 400, No of transactions perblock 2

Summary of performance metrics

Name	Succ	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
open	400	0	40.4	7.16	0.67	6.11	25.9

Resource monitor: docker

Name	CPU% (max)	CPU% (avg)	Memory(max) [MB]	Memory(avg) [MB]	Traffic In [MB]	Traffic Out [MB]	Disc Write [MB]	Disc Read [MB]
dev-peer0.org2.example.com-fabcar-1	4.17	1.59	17.8	11.90	1.12	0.467	0.00	1.76
dev-peer0.org1.example.com-fabcar-1	3.51	1.15	12.2	9.63	0.609	0.336	0.00	1.74
caliper	0.00	0.00	0.00	0.00	0.00	-	0.00	0.00
peer0.org1.example.com	31.54	15.12	142	142	2.91	3.72	1.46	2.24
ca.org2.example.com	0.00	0.00	5.79	5.75	0.000132	0.00	0.00	0.00
orderer.example.com	5.21	1.96	35.5	30.7	2.31	2.34	3.64	0.922
couchdb2	37.21	10.92	52.3	50.8	0.176	0.194	1.26	0.0664
couchdb1	37.68	15.67	54.1	52.9	0.239	0.266	1.57	0.0977
couchdb0	37.79	13.15	49.9	48.5	0.192	0.213	1.30	0.0664
ca.org1.example.com	0.00	0.00	16.5	16.3	0.000132	0.00	0.00	0.00
peer1.org2.example.com	10.21	5.56	66.8	56.7	1.96	0.394	2.55	2.10
orderer2.example.com	11.56	4.02	39.1	30.8	1.86	3.44	3.65	7.40
peer0.org2.example.com	34.76	15.14	89.0	84.3	2.19	3.19	1.72	0.582
orderer3.example.com	8.87	3.29	34.8	29.4	2.29	2.32	3.65	0.719
cli	0.00	0.00	2.95	2.94	0.000212	0.00	0.00	0.00
peer1.org1.example.com	22.83	10.15	81.8	76.4	1.94	0.333	2.36	0.102
couchdb3	35.93	15.26	66.7	65.2	0.329	0.295	1.70	0.109

Experiment 8 System: HLF 2.0, Total transactions 400, No of transactions perblock 2

Summary of performance metrics

Name	Succ	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
open	400	0	40.4	5.34	0.44	5.03	27.1

Resource monitor: docker

Name	CPU% (max)	CPU% (avg)	Memory(max) [MB]	Memory(avg) [MB]	Traffic In [MB]	Traffic Out [MB]	Disc Write [MB]	Disc Read [MB]
dev-peer0.org2.example.com-fabcar-1	3.49	1.13	12.5	9.47	0.712	0.347	0.00	1.86
dev-peer0.org1.example.com-fabcar-1	3.30	1.05	12.0	9.43	0.709	0.336	0.00	1.84
caliper	0.00	0.00	0.00	0.00	0.00	-	0.00	0.00
peer0.org1.example.com	32.28	15.27	105	102	2.87	3.42	1.90	2.04
ca.org2.example.com	0.00	0.00	5.79	5.65	0.000212	0.00	0.00	0.00
orderer.example.com	5.21	1.96	35.5	30.7	2.31	2.34	3.64	0.922
couchdb2	37.21	10.92	52.3	50.8	0.176	0.194	1.26	0.0664
couchdb1	37.68	15.37	54.1	52.7	0.239	0.266	1.57	0.0977
couchdb0	37.79	12.95	49.2	48.5	0.192	0.213	1.30	0.0664
ca.org1.example.com	0.00	0.00	15.5	15.3	0.000212	0.00	0.00	0.00
peer1.org2.example.com	10.62	5.56	66.8	59.7	1.96	0.394	2.55	2.10
orderer2.example.com	10.79	3.98	38.5	30.8	1.86	3.44	3.65	7.40
peer0.org2.example.com	34.76	15.84	90.9	85.3	2.89	3.19	1.72	0.582
orderer3.example.com	8.97	3.29	34.8	29.4	2.29	2.32	3.65	0.719
cli	0.00	0.00	2.85	2.84	0.000212	0.00	0.00	0.00
peer1.org1.example.com	22.83	10.15	81.8	76.4	1.94	0.333	2.36	0.102
couchdb3	37.93	16.26	46.7	45.2	0.259	0.295	1.70	0.109

Full factorial table to calculate the effects of various factors on throughput

I	A	B	C	AB	BC	CA	ABC	Y
1	1	1	1	1	1	1	1	24.4
1	-1	1	1	-1	1	-1	-1	26.7
1	1	-1	1	-1	-1	1	-1	24.6
1	-1	-1	1	1	-1	-1	1	26.9
1	1	1	-1	1	-1	-1	-1	25.4
1	-1	1	-1	-1	-1	1	1	26.9
1	1	-1	-1	-1	1	-1	1	25.9
1	-1	-1	-1	1	1	1	-1	27.1
207.9	7.3	1.9	2.7	0.3	0.3	1.9	0.3	
25.875	0.91	0.237	0.337	0.042	0.042	0.237	0.042	

Calculation SST=SSA+SSB+SSC+SSAB+SSBC+SSCA+SSABC
=
8x(0.91X0.91+0.237X0.237+0.337X0.337+0.042X0.042+0.042X0.042+0.237X0.237+0.042X0.042) = 6.624+0.45+0.45+1.41+1.41+0.45+1.41 =12.204

From Table we can observe that total variation explained by our A(Type of hyperledger version 1.4 or 1.6) = 6.624/12.204 = 55%

From Table we can observe that total variation explained by our B(No of transactions) = 0.45/12.204 = 3.6%

From Table we can observe that total variation explained by our B(No of transactions per block or block size) = 0.45/12.204 = 3.6%

From our observation we can say that System version have a major effect on throughput of HLF network.

No of transactions and transaction per-block have also a little effect on performance.

Full factorial table to calculate the effects of various factors on latency

I	A	B	C	AB	BC	CA	ABC	Z
1	1	1	1	1	1	1	1	6.21
1	-1	1	1	-1	1	-1	-1	5.53
1	1	-1	1	-1	-1	1	-1	6.11
1	-1	-1	1	1	-1	-1	1	6.02
1	1	1	-1	1	-1	-1	-1	6.03
1	-1	1	-1	-1	-1	1	1	5.53
1	1	-1	-1	-1	1	-1	1	6.11
1	-1	-1	-1	1	1	1	-1	5.03
46.57	2.35	0.03	1.17	0.01	0.81	0.81	1.17	
5.81	0.29	0.003	0.146	0.00125	0.101	0.101	0.146	

Calculation SST=SSA+SSB+SSC+SSAB+SSBC+SSCA+SSABC
= 0.672+0.00072+0.170+0.000125+0.081+0.081+0.170 =1.175

From Table we can observe that total variation explained by our A(Type of hyperledger version 1.4 or 1.6) = 0.672/1.175 = 57%

From Table we can observe that total variation explained by our B(No of transactions) = 0.00072/1.175 = 0.06%

From Table we can observe that total variation explained by our B(No of transactions per block or block size) = 0.170/1.175=14.5%

From our observation we can say that System version have a major effect on latency of HLF network.

No of transaction per-block have also have a major effect on latency of system.

From both of our observation we can infer that HLF 2.0 is a better choice when we consider building a blockchain network, also increasing no of transaction per block helps in decreasing our latency.

Since There are many factors like type of endorsing, no of peers, state database whose study on performance of HLF Blockchain network is limited due to inefficient resource and team.
And i feel that my report on performance analysis of HLF blockchain network is incomplete without them so, now I will summarise some of the research papers in this area.

Summarisation of some of the relevant work on our topic of performance evaluation of Blockchain network

Qassim Nasir et al [1] in their analysis used HLF network of v0.6 and v1.0 study used HPC server with the Intel(R) Xeon(R) CPU E5-2690, 2.60 GHz, 24 core CPU, 64 GB RAM, and running Ubuntu 16.04.

Thakkar et al [2] in their extensive analysis of HLF v1.0 Network used 4 organisations each with 2 endorsing peers for a total of 8 peer nodes, one order with Kafka-zookeeper cluster backing it. All nodes and kafka-zookeeper run on IBM Softlayer Datacenter. Each vNodes run on 32 vCPUs of Intel Xeon(R) CPU with 32 GB of memory. 3 powerful client machines were used to generate the workload their system configuration was 56 vCPUs with 128GB of memory. Node were connected to 3gbps Datacenter Network. In the absence of any standard Benchmark author and their team also developed their own benchmark. This benchmark only consider micro benchmarking of HLF network. Figure below shows the experiment setup of HLF Network in this study

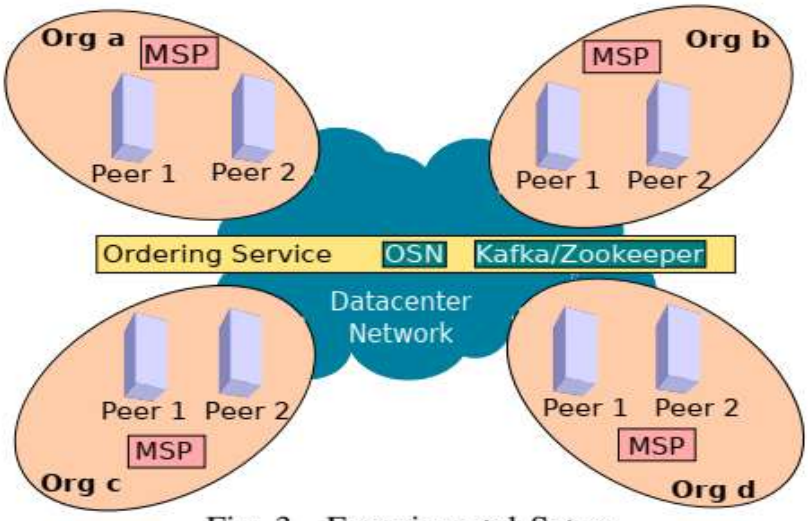


Fig. 3. Experimental Setup

Table Below shows the configuration of various paraments in under the study

TABLE I
DEFAULT CONFIGURATION FOR ALL EXPERIMENTS UNLESS SPECIFIED OTHERWISE.

Parameters	Values
Number of Channels	1
Transaction Complexity	1 KV write (1-w) of size 20 bytes
StateDB Database	GoLevelDB
Peer Resources	32 vCPUs, 3 Gbps link
Endorsement Policy AND/OR	OR [AND(a ,b, c), AND(a, b, d), AND(b, c, d), AND(a, c, d)]
Block Size	30 transactions per block
Block Timeout	1 second

Salma Shalaby et al [3] in their study used HLF v1.4 to build the health network, this network had 1 channel that connected org1 and org2 as the peer organisation and one orderer organisation. Each peer organisation has 2 connected peers in turn. Solo-based desing was adopted for the study purpose.

Takuya Nakaike et al [4] have concentrated their performance analysis to performance bottleneck that is incurred by the database systems used to store ledgers
for their study team have developers a Hyperledger Fabric GoLevelDB (HLF-GLDB) benchmark, which characterizes the performance of database access in Hyperledger Fabric via simulating the transaction behaviors.

Suporn Pongnumkul et al [5] have performed a comparative study of HLF and Ethereum. Experiments are carried on is Amazon AWS EC2 (c4.2xlarge instance) with the Intel E5-1650 8 core CPU, 15GB RAM, 128GB SSD hard drive and running Ubuntu 16.04.For Ethereum they used Geth 1.5.8 and for HLF they used v0.6 to deploy network.

V. EXPERIMENTAL RESULTS

A. Impact of Transaction Arrival Rate and Block Size:

Thakkar et al [2] carried more that 1000 transaction to find the effect of varying tps on performance. team used following configuration to find the effect of transactions arrival rate and block size on latency and throughput

CONFIGURATION TO IDENTIFY THE IMPACT OF BLOCK SIZE AND TRANSACTION ARRIVAL RATE.

Parameters	Values
Tx. Arrival Rate	25, 50, 75, 100, 125, 150, 175 (tps)
Block Size	10, 30, 50, 100 (#tx)

Following graph illustrate the results of their finding

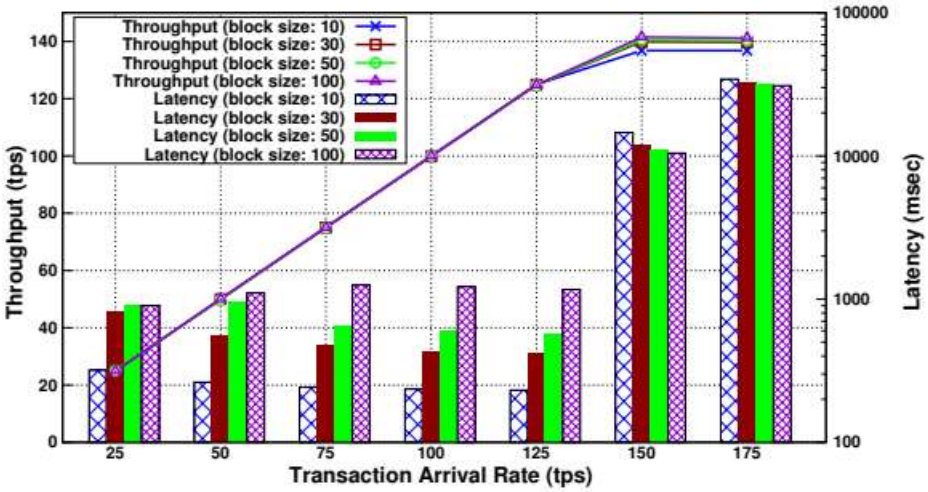


Fig. 4. Impact of the block size and transaction arrival rate on performance.

We can observe that transaction rates flattens at around 140tps which can be considered saturation point for the HLF v1.0 Configuration, we can also observe increase in the latency from 100ms to 10s. This latency majorly contributed by VSCC validation latency

Based on observation some optimisation were suggested by the authors

* For transaction rate lower than saturation point to achieve lower latency use lower block size

* use large block size when high tps is considered to reduce the latency and maximise the throughput

B. Impact of Endorsement policy

Thakkar et al [2] studied the effect of different endorsement policies over the performance of blockchain, effect of various policies like AND/OR and Not were studied at varying transaction per second. Following table illustrate the experimental setup to study the effect of endorsement policy

TABLE III
CONFIGURATION TO IDENTIFY THE IMPACT OF ENDORSEMENT POLICIES.

Parameters	Values
Endorsement Policy (AND/OR)	1) OR [a, b, c, d] 2) OR [AND(a ,b), AND(a, c), AND(a, d), AND(b, c), AND(b, d), AND(C, D)] 3) OR [AND(a ,b, c), AND(a, b, d), AND(b, c, d), AND(a, c, d)] 4) AND [a, b, c, d]
Endorsement Policy (NOutOf)	1) 1-OutOf [a ,b ,c, d] 2) 2-OutOf [a ,b ,c, d] 3) 3-OutOf [a ,b ,c, d] 4) 4-OutOf [a ,b ,c, d]
Tx. Arrival Rate	125, 150, 175 (tps)

Based on the experimental setup authors have observed following results that is illustrated in the graph below

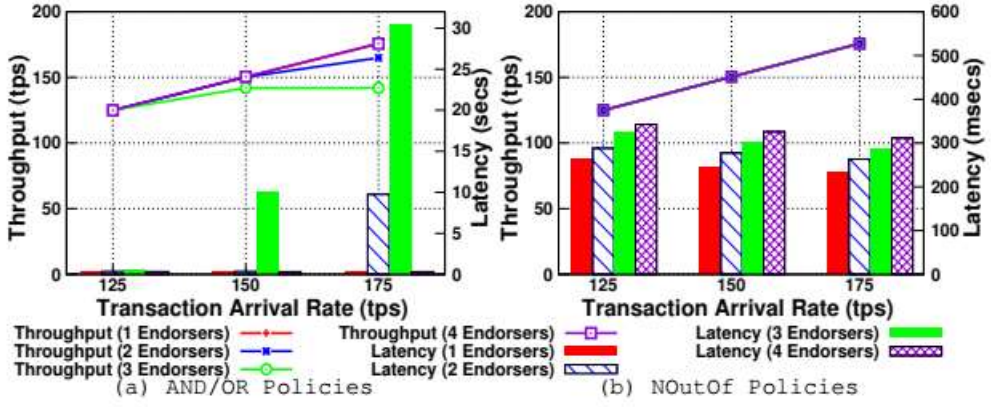


Fig. 6. Impact of different endorsement policies (AND/OR and NOutOf).

we can observe from the setup that when we have no or as in case of 1 and 4 with AND and OR with fewer sub policies system performance well, this can be due to fewer signature verification. This observation can be further verified by

the results of 2nd and 3rd where with additional sub policies we observe increase in the latency can be observed from 68ms to 137ms.

* Optimisation of system performance

To keep throughput high and latency low, define policies with fewer sub policies and signatures

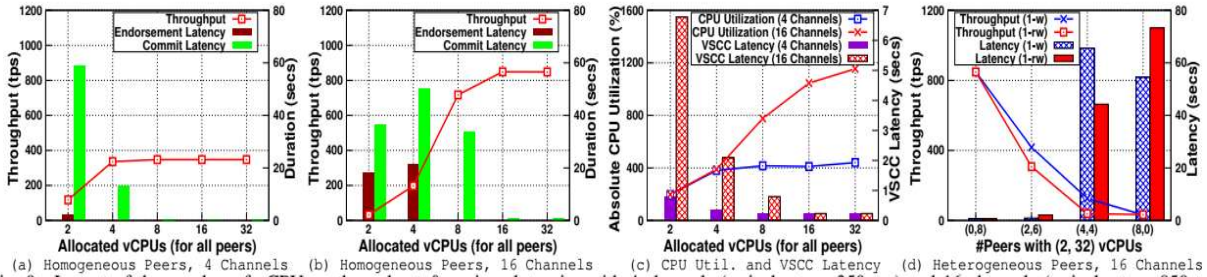
C . Impact of Channels and Resource Allocation

Thakkar et al [2] studied effect of heterogeneous resourse setup with 4 and 16 channels. The experimental setup used under study is illustrated in the table below

TABLE VI
CONFIGURATION TO IDENTIFY THE IMPACT OF HETEROGENEOUS SETUP.

Parameters	Values
Number of Channels	16
Transaction Complexity	1 KV write (1-w) & 1 KV read/write (1-rw)
#Peers with (2, 32) vCPUs	(0, 8), (2, 4), (4, 4), (8, 0) peers
Tx. Arrival Rate	850 tps

studies can we well understood with graph generated by the observance of experimental data



(a) Homogeneous Peers, 4 Channels (b) Homogeneous Peers, 16 Channels (c) CPU Util. and VSCC Latency (d) Heterogeneous Peers, 16 Channels

Fig. 9. Impact of the number of vCPU on throughput & various latencies with 4 channels (arrival rate = 350 tps) and 16 channels (arrival rate = 850 tps).

Observation can be summarised as below:

* With the increase in the no of channels throughput increased and latency decreased

* at moderated setup when number vCPUs were less than the no of channels performance decrease

optimisation suggestions

* To achieve better performance allocate at least one vCPUs to each channel

* avoid heterogeneous setup as performance will be determined by the poorest performing peer.

4 Impact of Ledger Database

* Fabric performance was 3 times better that the implementation using couch database.

* CouchDB endorsement latency and ledger update latency increase with increase in the no of writes per transactions

Optimisation suggestion

For better performance use GoLevelDB but if rich-query support for read-only transaction is important then use CouchDB.

Takuya Nakaike et al, studied the performance characterisation using a GOLevelDB Benchmark. This benchmark has eight configurable parameters: BlockSize, NumBlocks, ValueSize, NumReads, NumWrites, StateDBSize, Compression, and BloomFilter.

Configuration code for this setup is snipped below

```
for i := 0; i < number_of_blocks; i++ {
validateReadSet()
writeBlock()
```

checkTransactionIDs()
updateIndexDB()
updateStateDB()
updateHistoryDB()
}

Under their study of HLF network TPS of Hyperledger Fabric measured by HLF-GLDB with the following parameters.

- BlockSize: 2 MB
- NumBlocks: 4000
- ValueSize: 256 or 1024 bytes
- NumReads: 2
- NumWrites: 2
- StateDBSize: 256 or 1024 MB
- Compression: true or false
- BloomFilter: 30 or 0

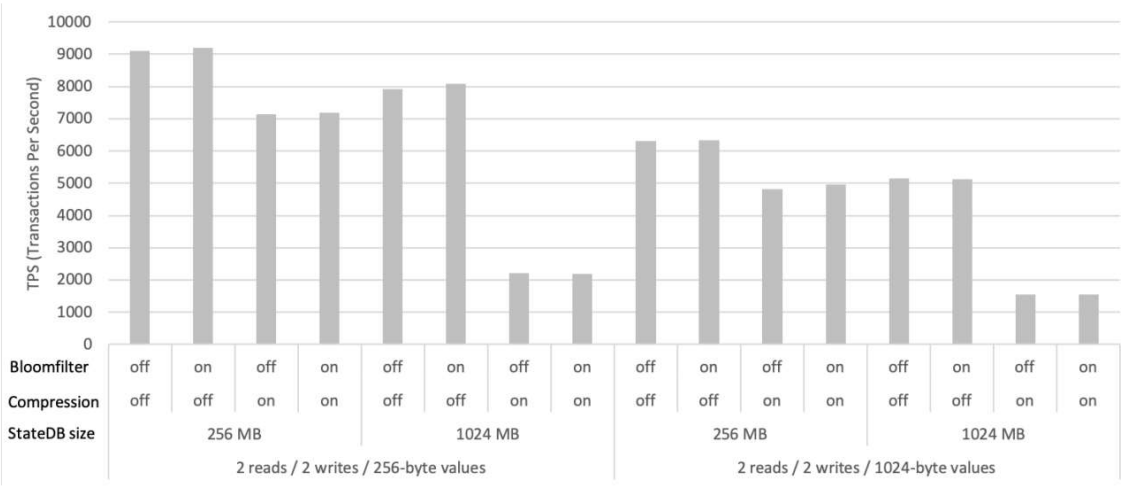


Fig. 6. Simulated transactions per second (TPS) of Hyperledger Fabric via HLF-GLDB with two reads and two writes in a transaction.

Results from their observation is summarised under this grap
Summarisation of various findings from their study:

* While compression of GoLevelDB is very effective way to reduce the cost of storage but cost of compression significantly reduces the throughput and increase the latency, so it would be recommanded to turn off the compression of stateDB

* StateDB size also have significant over HLF performance reducing the size of StateDB have a positive effect in increasing the performance of HLF, so its suggested to develop chaincode with less key storage in StateDB.

* Reducing no of read writes in each transaction have significant improvement over the performance, so merging multiple key value into single one on our chaincode will improve our HLF network performance.

Salma Shalaby et al, used an organisational setup of healthcare system to study the performance of HLF Network a graphical illustration of their setup and configuration is shown below

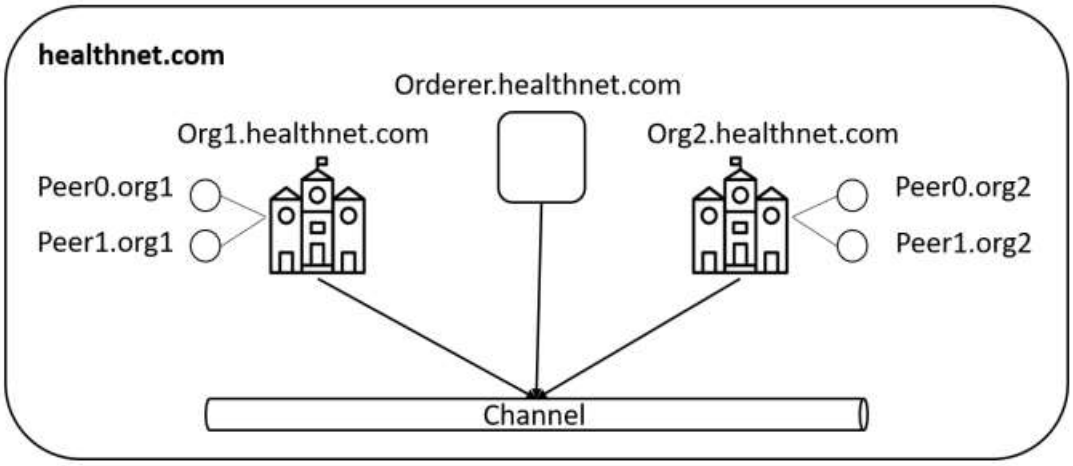


Fig. 3. : High-level architecture of the network.

Configuration setup for above experimental setup

TABLE III
EXPERIMENTAL SETTINGS

	Batch-timeout	Batch Size (Max. Transaction Count)	Number of Endorsers
Exp. 1	2 Seconds	45 Tx	4 endorsers
Exp. 2	5 Seconds	45 Tx	4 endorsers
Exp. 3	8 Seconds	45 Tx	4 endorsers
Exp. 4	200 Seconds	2 Tx	4 endorsers
Exp. 5	200 Seconds	5 Tx	4 endorsers
Exp. 6	200 Seconds	10 Tx	4 endorsers
Exp. 7	2 Seconds	45 Tx	2 endorsers

Findings and results form their study is summarised as below :

* Effect of changing batch-timeout:

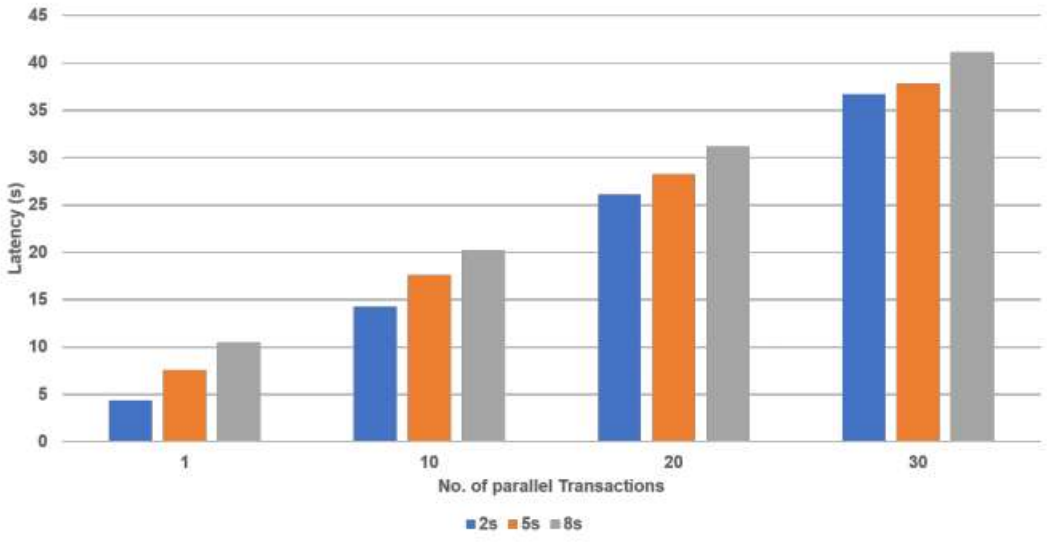


Fig. 4. Average latency with varying Batch-timeout

We can observe from above graph that end to end performance is affected by changing batch time out also the no of parallel transaction tends to decrease the performance to a threshold.

* Effect of Batch size:

Although with our previous experiment we found that performance decrease with the increase in the no of parallel transaction but when we increase the block size this effect subside and latency tends to decrease. This is due to the fact that there will be less no of blocks and higher no of transaction per block.

* Effect of Endorsers

From experiment 1 and 7 we see that with increase in no of endorsers latency increase which leads in degradation of performance, the reason behind this is because peers have to wait for endorsers for finalising each transaction.

Observation of HLF performance by Qassim Nasir et al:

Their team observed the performance of HLF 0.6 vs HLF 1.0, although is quite obvious that a new version would have some improvements over the older one. Results by authors is quite interesting, they have compared performance of both version with varying workload and no of nodes. It was found that overall performance over all evaluation matrices
Execution time, throughput and latency is much better that HLF 0.6. HLF 1.0 maintains its same performance over all performance matrices while HLF06 varies it this case.
With only single channel HLF 0.6 have somewhat identical performance with HLF 1.o.

Observation of HLF performance by Suporn Pongnumkul et al:

In this work authors have observed the performance of HLF in comparison to Ethereum. Their findings can be summarised below

It was observed that HLF performs better than Ethereum when workload varied up to 10,000 transactions. This difference in performance is more significant in terms of latency and throughput. With similar computation resource Ethereum performs better at concurrent transactions. Although authors have not considered effect of endorsing peers in the overall performance.

Conclusion

From the results of works of various researchers we have seen that performance of Fabric network is greatly affected by no of tps, No of channels, type of database used for State database, endorsing policy used.
We seen that HLF outperforms Ethereum blockchain network, and in further studies we noticed that Successive HLF versions outperforms its predecessors network versions. This result is further verified by the results of my own project work, that HLF network 2.0 outperforms HLF network 1.4.

REFERENCES

[1] Qassim Nasir , Ilham A. Qasse, Manar Abu Talib, and Ali Bou Nassif, " Performance Analysis of Hyperledger Fabric Platforms"

[2] Parth Thakkar, Senthil Nathan, N Balaji Viswanathan, "Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform"

[3] Salma Shalaby, Alaa Awad Abdellatif*y*, Abdulla Al-Ali, Amr Mohamed, Aiman Erbad, Mohsen Guizani , " Performance Evaluation of Hyperledger Fabric"

[4]Takuya Nakaike, Qi Zhang, Yohei Ueda, Tatsushi Inagaki, Moriyoshi Ohara, " Hyperledger Fabric Performance Characterization and Optimization Using GoLevelDB Benchmark"

[5]Suporn Pongnumkul, Chaiyaphum Siripanpornchana, and Suttipong Thajchayapong, "Performance Analysis of Private Blockchain Platforms in Varying Workloads"