1. 개요

본 문서는 "Caliper 를 이용한 Hyperledger Fabric 성능 측정"을 위한 가이드 문서이다. Caliper 구동을 위한 개발환경 설치 가이드와 기타 자료를 포함하고 있으며 다음 링크를 기반으로 작성되었다. "https://hyperledger.github.io/caliper/docs/1_Getting_Started.html"

Caliper: Hyperledger 에서 제공하는 블록체인 성능 측정 Framework.

Caliper 를 통해 다음 항목을 측정할 수 있다.

- Success rate
- Transaction/Read throughput
- Transaction/Read latency(minumum, maximum, average, percentile)
- Resource consumption(CPU, Memory, Network IO,…)

2. Architecture

"https://hyperledger.github.io/caliper/docs/2_Architecture.html" 참고

3. Pre-requisties

- NodeJS 8.X
- Node-gyp
- Docker
- Docker-compose

4. Caliper 설치 및 환경구성 (Fabric v1.4)

1. **원하는 경로에 Caliper 클론 (설치된 경로는 이하 $CALIPER_HOME)**

```
git clone https://github.com/hyperledger/caliper.git
```

2. **Install base dependencies**

```
cd $CALIPER_HOME
npm install
```

3. **All packages clean**

```
npm run repoclean
```

4. **Install all package dependencies and link any cross dependencies**

```
npm run bootstrap
```

>> Caliper 는 nodejs 기반으로 작동된다.
따라서, 필요 모듈들을 npm 을 통해서 설치한다. (2)
npm run "NAME"을 통해 현재 경로의 package.json → scripts → "NAME" 에 해당하는 명령어를 실행한다. (3, 4)
3, 4 과정에서는 lerna 를 사용하는데, bootstrap(4)과정에서 오류 발생 시 수동으로 dependencies 를 설치해준다. (5)
4 번을 수행하는데에 문제가 없다면 5, 6, 7 번 생략

5. **Install all package dependencies and link any cross dependencies (handwork)**

```
npm install --save-prod fabric-ca-client@1.4.0 fabric-client@1.4.0 fabric-network@1.4.0 \
grpc@1.10.1 fabric-protos@2.0.0-snapshot.1
```

6. **Install caliper-core dependencies**

```
cd $CALIPER_HOME/packages/caliper-core
npm install
```

7. **Copy the dependencies packages**

```
cd $CALIPER_HOME/packages/
cp -r ./caliper-core ./caliper-fabric ./caliper-fabric-ccp ../node_modules/
```

Caliper 구동에는 benchmark 파일과 network 파일이 필요하다.
benchmark: 어떠한 부하를 얼마나, 어떻게 발생시킬지 등을 설정
network: benchmark 의 대상이 되는 네트워크 구성을 설정

4.1 Caliper benchmark ($CALIPER_HOME/packages/caliper-application/benchmark)

```yaml
---
test:
  name: simple
  description: This is an example benchmark for caliper, to test the backe
    performance with simple account opening & querying transactions
  clients:
    type: local
    number: 1
  rounds:
  - label: open
    description: Test description for the opening of an account through th
    txNumber:
    - 100
    rateControl:
    - type: fixed-rate
      opts:
        tps: 50
    arguments:
      money: 10000
    callback: benchmark/simple/open.js
  - label: query
    description: Test description for the query performance of the deploye
    txNumber:
    - 100
    rateControl:
    - type: fixed-rate
      opts:
        tps: 100
    callback: benchmark/simple/query.js
monitor:
  type:
  - docker
  - process
  docker:
    name:
    - all
  process:
  - command: node
    arguments: local-client.js
    multiOutput: avg
  interval: 1
```

가장 기본적인 benckmark 파일이다. (config.yaml)
다양한 benchmark option 중 주요 항목들만 다룰 것이다.
(더욱 자세한 사항은 공식 홈페이지를 참조 https://hyperledger.github.io/caliper/docs/2_Architecture.html)

test: name: → 테스트의 이름
test: clients: type: → 클라이언트의 구성 (local/zookeeper)
test: clients: number: → type 이 local 일 경우, txn 을 발생시킬 client 의 수
test: rounds: label: → txn 의 이름. network file 의 context 와 반드시 매핑되어야 함
test: rounds: txNumber: → round 당 발생시킬 총 txn 의 양 (복수 설정 가능)
test: rounds: rateControl: type: → txn 발생 속도 정의
test: rounds: rateControl: opts: tps: → round 에서 초당 발생하는 txn 의 양
test: rounds: arguments: → txn 에 포함시킬 arguments (key, value 사용자 정의)
test: rounds: calleback: → 테스트에서 사용될 사용자 정의 모듈
monitor: docker: name: → caliper 에서 monitoring 을 할 container 정의

4.2 Caliper network ($CALIPER_HOME/packages/caliper-application/network)

Benchmarking 의 대상이 되는 network config 파일이다.
network 에서 txn 을 처리하기 위해 필요한 crypto, org, channel, chaincode, endorsment-policy, grpc 등의 설정 정보를 포함하고 있다.

Caliper 에서는 fabric 과 fabric-ccp 모드가 존재한다.
fabric-ccp 는 caliper 의 fabric 모드에서는 지원하지 않던 하나 이상의 orderer, 하나 이상의 channel 등을 지원한다.
fabric: *.json
fabric-ccp : *.yaml
편리하게도 Caliper 자체적으로 fabric-v1.x, XorgXpeerXdb 와 같이 다양한 네트워크를 지원한다.
(더욱 자세한 사항은 공식 홈페이지를 참조
fabric: https://hyperledger.github.io/caliper/docs/Fabric_Configuration.html
fabric-ccp: https://hyperledger.github.io/caliper/docs/Fabric_Ccp_Configuration.html)

5. Caliper 실행

Caliper 구동을 위해 nodejs 를 이용하여
$CALIPER_HOME/packages/caliper-application/scripts/run-benchmark.js 를 실행한다.}

```
cd $CALIPER_HOME/packages/caliper-application/scripts
node run-benchmark.js -c myconfig -n mynetwork
```

-c : benchmark 의 config 파일
-n : network 의 config 파일

caliper 를 실행하면 network config 파일 내부 caliper: command: start, end 에 해당하는 명령어를 실행한다.
start: 동일 경로의 docker-compose.yaml 을 참조하여 container 들을 올린다.
end: container 들을 내리고, rmi 까지 수행한다.
자신만의 네트워크에서 성능 측정을 하고자 한다면 해당 명령어들을 필히 확인하고 수정 혹은 삭제하는 것을 권장한다.

Caliper flow:

1) Start the containers
2) Create channel
3) Join channel (all peers)
4) Install chaincode (all chaincodes, all peers)
5) Instantiate chaincode (all chaincodes, all peers)
6) Test ---------- **monitoring & report**
7) Stop container

Caliper 에서는 다양한 sample chaincode, sample network 를 제공한다.
본 문서에서는 총 4 가지의 상황을 다룰 것이다.
1. Sample Network & Sample Chaincode
2. Sample Network & Own Chaincode
3. Own Network & Own Chaincode
4. Own Network & Own Chaincode & Distributed Client

5.1 Sample Network & Sample Chaincode

```
cd $CALIPER_HOME/packages/caliper-application/scripts
node run-benchmark.js -c ../benchmark/simple/config.yaml -n ../network/fabric-v1.4/2org1peercouchdb/fabric-go.json
```

Test	Name	Succ	Fail	Send Rate	Max Latency	Min Latency	Avg Latency	Throughput
1	open	100	0	50.5 tps	1.10 s	0.45 s	0.74 s	40.9 tps
2	query	100	0	101.0 tps	0.26 s	0.04 s	0.13 s	95.2 tps

```
info: [monitor.js]:
 ### resource stats (maximum) ###
info: [monitor.js]:
```

TYPE	NAME	Memory(max)	CPU(max)	Traffic In	Traffic Out	Disc Read	Disc Write
Process	node local-client.js(avg)	-	NaN%	-	-	-	-
Docker	dev-peer0.org1.example.co...nk-v0	5.2MB	0.00%	70B	0B	0B	0B
Docker	dev-peer0.org2.example.co...nk-v0	6.1MB	0.00%	70B	0B	0B	0B
Docker	dev-peer0.org1.example.co...le-v0	5.3MB	2.74%	164.4KB	61.6KB	112.0KB	0B
Docker	dev-peer0.org2.example.co...le-v0	5.4MB	2.80%	168.0KB	63.9KB	112.0KB	0B
Docker	dev-peer0.org2.example.co...rm-v0	5.1MB	0.00%	0B	0B	0B	0B
Docker	dev-peer0.org1.example.co...rm-v0	6.1MB	0.00%	0B	0B	0B	0B
Docker	dev-peer0.org2.example.co...es-v0	5.1MB	0.00%	0B	0B	0B	0B
Docker	dev-peer0.org1.example.co...es-v0	5.3MB	0.00%	0B	0B	0B	0B
Docker	peer0.org2.example.com	136.3MB	17.40%	685.5KB	379.8KB	1.7MB	1.3MB
Docker	peer0.org1.example.com	186.9MB	19.63%	696.2KB	509.1KB	3.2MB	1.3MB
Docker	couchdb.org1.example.com	97.6MB	22.66%	83.4KB	96.3KB	48.0KB	592.0KB
Docker	orderer.example.com	24.1MB	3.98%	409.0KB	817.8KB	3.5MB	960.0KB
Docker	ca.org1.example.com	7.2MB	0.00%	0B	0B	0B	0B
Docker	couchdb.org2.example.com	95.2MB	25.99%	79.2KB	93.9KB	104.0KB	544.0KB
Docker	ca.org2.example.com	7.6MB	0.00%	0B	0B	0B	0B

위와 같이 txn 들의 success, fail 횟수, tps 와 container 들의 resource 사용량 들을 출력한다.
(위 결과는 monitor: docker: name: all 로 설정되어 있어, 현재 local 에 실행중인 container 들의 resource 사용 정보가 기록된다.
사용하지 않더라도 network config 에 존재하는 모든 chaincode 가 install&instantiate 된다.)

5.2 Sample Network & Own Chaincode

다음은 caliper 를 통해 원하는 chaincode 를 테스트 하기위한 환경구성이다.

- **Edit network config**

```
"chaincodes": [
  {"id": "waetcc", "path": "contract/fabric/waet", "language":"golang", "version": "v0", "channel": "mychannel"}
],
  "context": {
    "mytest": "mychannel"
  }
```

chiancode 의 정보를 입력하고, context 입력을 통해 txn 이름을 설정한다.
chaincode name(id)와 version 그리고 context 값은 추후 benchmark config 에 사용된다.

- **Benchmark config 파일 수정 (좌: benchmark config, 우: module)**

```yaml
test:
  name: my
  clients:
    type: local
    number: 1
  rounds:
  - label: mytest
    txNumber:
    - 100
    rateControl:
    - type: fixed-rate
      opts:
        tps: 50
    arguments:
      type: 'TKN'
      start: '1'
      amount: '100'
    callback: benchmark/simple/mymodule.js
monitor:
  type:
  - docker
  - process
  docker:
    name:
    - all
  process:
  - command: node
    arguments: local-client.js
    multiOutput: avg
  interval: 1
```

```javascript
'use strict';


let type, start, amount;
let bc, contx;
module.exports.init = function(blockchain, context, args) {

    type = args.type;
    start = args.start;
    amount = args.amount;

    bc = blockchain;
    contx = context;

    return Promise.resolve();
};

function generateWorkload() {
    let workload = [];

     workload.push({
         'verb': 'batch',
         'type': type,
         'start': start,
         'amount': amount
     });

    return workload;
}

module.exports.run = function() {
    let args = generateWorkload();
    return bc.invokeSmartContract(contx, 'waetcc', 'v0', args, 100);
};

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

----------benchmark config file ----------
label: → network config 의 context 와 동일하게 입력
txNumber:, rateControl: → 원하는 부하 입력
argument: → argument 가 필요하면 입력
callback: → test 를 진행할 module 의 경로를 입력
----------module(mymodule.js)----------
chaincode 에 부합하는 argument 를 포함하여
bc.invokeSmartContract()를 호출한다.
query: bc.querySmartContract()

 # network config 에서 설정한 chaincode name 과 version 이 동일해야 한다.

```
cd $CALIPER_HOME/packages/caliper-application/scripts
node run-benchmark.js -c mybenchmark -n ../network/fabric-v1.4/2org1peercouchdb/fabric-go.json
```

----------report----------

```
------+--------+------+------+-----------+-------------+-------------+-------------+-----------+
 Test | Name   | Succ | Fail | Send Rate | Max Latency | Min Latency | Avg Latency | Throughput |
------|--------|------|------|-----------|-------------|-------------|-------------|-----------|
 1    | mytest | 100  | 0    | 50.4 tps  | 0.39 s      | 0.08 s      | 0.21 s      | 48.4 tps   |
------+--------+------+------+-----------+-------------+-------------+-------------+-----------+

info: [monitor.js]:
### resource stats (maximum) ###
info: [monitor.js]:
---------+---------------------------------+-------------+---------+------------+-------------+----------+------------+
 TYPE    | NAME                            | Memory(max) | CPU(max) | Traffic In | Traffic Out | Disc Read | Disc Write |
---------|---------------------------------|-------------|---------|------------|-------------|----------|------------|
 Process | node local-client.js(avg)       | -           | NaN%    | -          | -           | -        | -          |
---------|---------------------------------|-------------|---------|------------|-------------|----------|------------|
 Docker  | dev-peer0.org1.example.co...cc-v0 | 5.4MB       | 2.69%   | 85.9KB     | 18.5KB      | 0B       | 0B         |
---------|---------------------------------|-------------|---------|------------|-------------|----------|------------|
 Docker  | dev-peer0.org2.example.co...cc-v0 | 5.2MB       | 2.76%   | 86.8KB     | 18.7KB      | 0B       | 0B         |
---------|---------------------------------|-------------|---------|------------|-------------|----------|------------|
 Docker  | peer0.org1.example.com          | 111.7MB     | 17.85%  | 354.7KB    | 215.1KB     | 0B       | 736.0KB    |
---------|---------------------------------|-------------|---------|------------|-------------|----------|------------|
 Docker  | peer0.org2.example.com          | 66.9MB      | 17.60%  | 351.2KB    | 175.8KB     | 0B       | 736.0KB    |
---------|---------------------------------|-------------|---------|------------|-------------|----------|------------|
 Docker  | ca.org2.example.com             | 7.4MB       | 0.00%   | 0B         | 0B          | 0B       | 0B         |
---------|---------------------------------|-------------|---------|------------|-------------|----------|------------|
 Docker  | couchdb.org1.example.com        | 90.2MB      | 9.61%   | 6.9KB      | 10.8KB      | 0B       | 24.0KB     |
---------|---------------------------------|-------------|---------|------------|-------------|----------|------------|
 Docker  | couchdb.org2.example.com        | 89.6MB      | 8.05%   | 7.1KB      | 10.8KB      | 0B       | 8.0KB      |
---------|---------------------------------|-------------|---------|------------|-------------|----------|------------|
 Docker  | ca.org1.example.com             | 7.5MB       | 0.00%   | 0B         | 0B          | 0B       | 0B         |
---------|---------------------------------|-------------|---------|------------|-------------|----------|------------|
 Docker  | orderer.example.com             | 14.6MB      | 8.36%   | 262.7KB    | 490.4KB     | 0B       | 560.0KB    |
---------+---------------------------------+-------------+---------+------------+-------------+----------+------------+
```

문제없이 작동하는 모습을 확인할 수 있다.

5.3 Own Network & Own Chaincode

다음은 caliper 를 통해 원하는 chaincode 와 network 의 성능을 테스트 하기위한 환경구성이다.
본 상황에서는 Single Node 환경을 생략하고, 바로 Multiple Node 환경에서 시작한다.
Multiple Node 환경을 위해 수동으로 container 들을 실행한다.

Own Network 는
node1 : peer0.org1, peer1.org1, orderer1, orderer2, zoo1, zoo2, kafka1, kafka2
node2 : peer0.org2, peer1.org2, orderer3, zoo3, kafka3, kafka4
로 구성되어 있다. (caliper 는 node2 에서 작동할 것이다.)

caliper 의 'fabric'에서는 multi orderer 환경을 제공하지 않기 때문에 'fabric-ccp'를 사용한다.
'fabric-ccp'의 network config 와 'fabric'의 network config 는 차이가 있다.
이미 제공되어 있는 network config 를 참고하고, 수정을 통해 caliper 에 own network 환경을 구성한다. (network config path: $CAHLIPER_HOME/packages/caliper-application/network/ ...)

입력이 필요한 정보는 다음과 같다.
----------network config----------

- **clients:**
  txn submit 을 위한 client's id

```
clients:
  client0.org1.example.com:
    client:
      organization: Org1
      credentialStore:
        path: /tmp/hfc-kvs/org1
        cryptoStore:
          path: /tmp/hfc-cvs/org1
      clientPrivateKey:
        path: network/fabric-v1.4/config-lgcns-2org2peer3orderer/crypto-config/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp/keystore/ea983f51466ca88de01192f40448bfe81107277c410a5a1cc72a2f4d63b67bd2_sk
      clientSignedCert:
        path: network/fabric-v1.4/config-lgcns-2org2peer3orderer/crypto-config/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp/signcerts/User1@org1.example.com-cert.pem

  client0.org2.example.com:
    client:
      organization: Org2
      credentialStore:
        path: /tmp/hfc-kvs/org2
        cryptoStore:
          path: /tmp/hfc-cvs/org2
      clientPrivateKey:
        path: network/fabric-v1.4/config-lgcns-2org2peer3orderer/crypto-config/peerOrganizations/org2.example.com/users/User1@org2.example.com/msp/keystore/e99902a4e59cff172d82b4222ea4d819421a5c7ea5761c93b58cfbdaf94d6c51_sk
      clientSignedCert:
        path: network/fabric-v1.4/config-lgcns-2org2peer3orderer/crypto-config/peerOrganizations/org2.example.com/users/User1@org2.example.com/msp/signcerts/User1@org2.example.com-cert.pem
```

- **channel:**
  테스트가 필요한 channel 들의 정보.
  - channel.tx (caliper 에서 create channel 을 하기 위함)
  - all orderer
  - all peer
  - all chaincode (caliper 에서 install & instantiate 를 하기 위함)

```
channels:
  mychannel:
    configBinary: network/fabric-v1.4/config-lgcns-2org2peer3orderer/channel.tx
    created: false
    orderers:
    - orderer1.example.com
    - orderer2.example.com
    - orderer3.example.com
    peers:
      peer0.org1.example.com:
        eventSource: true
      peer1.org1.example.com:
        eventSource: true
      peer0.org2.example.com:
        eventSource: true
      peer1.org2.example.com:
        eventSource: true

    chaincodes:
    - id: waetcc
      version: v0
      language: golang
      path: contract/fabric/lgcns/waetcc
    - id: setlcc
      version: v0
      language: golang
      path: contract/fabric/lgcns/setlcc
    - id: simple
      version: v0
      language: golang
      path: contract/fabric/simple/go
```

- **organiztaions:**
  Org 별 msp 정보

```
organizations:
  Org1:
    mspid: Org1MSP
    peers:
    - peer0.org1.example.com
    - peer1.org1.example.com
    adminPrivateKey:
      path: network/fabric-v1.4/config-lgcns-2org2peer3orderer/crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/keystore/a31c729996e79381ab69025b3055224712a087e0584f3b3a4f4553eef2c78a52_sk
    signedCert:
      path: network/fabric-v1.4/config-lgcns-2org2peer3orderer/crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/signcerts/Admin@org1.example.com-cert.pem

  Org2:
    mspid: Org2MSP
    peers:
    - peer0.org2.example.com
```

- **orderers: peers:**
  orderers, peers 의 grpc 정보

```
orderers:
  orderer1.example.com:
    url: grpc://192.168.56.2:7050
    grpcOptions:
      ssl-target-name-override: orderer1.example.com
  orderer2.example.com:
    url: grpc://192.168.56.2:8050
    grpcOptions:
      ssl-target-name-override: orderer2.example.com
  orderer3.example.com:
    url: grpc://192.168.56.3:9050
    grpcOptions:
      ssl-target-name-override: orderer3.example.com

peers:
  peer0.org1.example.com:
    url: grpc://192.168.56.2:7051
    grpcOptions:
      ssl-target-name-override: peer0.org1.example.com
      grpc.keepalive_time_ms: 600000

  peer1.org1.example.com:
    url: grpc://192.168.56.2:8051
    grpcOptions:
      ssl-target-name-override: peer1.org1.example.com
      grpc.keepalive_time_ms: 600000

  peer0.org2.example.com:
    url: grpc://localhost:9051
    grpcOptions:
      ssl-target-name-override: peer0.org2.example.com
      grpc.keepalive_time_ms: 600000

  peer1.org2.example.com:
    url: grpc://localhost:10051
    grpcOptions:
      ssl-target-name-override: peer1.org2.example.com
      grpc.keepalive_time_ms: 600000
```

- **command:**
  마지막으로 수동으로 network 를 띄우기 때문에 caliper-flow 1 번을 수행하지 않아야 한다.

```
caliper:
  blockchain: fabric-ccp
  command:
    start: sleep 2s
    end: sleep 2s
```

  위와 같이 start command 를 수정하고, log 를 보기위해 end command 도 수정한 모습이다.

----------benchmark config----------

5.2 와 동일 파일 사용

----------module----------

fabric-ccp 일 때 module 에서 호출하는 invokeSmartContract()메소드가 필요로 하는 인자의 형식이 달라진다.
fabric 과 fabric-ccp 모두 동일한 module 을 사용하기 위해 mymodule.js 를 수정한다.

```
function generateWorkload() {
    let workload = [];

    workload.push({
        'verb': 'batch',
        'type': type,
        'start': start,
        'amount': amount
    });

    return workload;
}
```

변경 전

```
function generateWorkload() {
    let workload = [];

    if (bc.bcType === 'fabric-ccp') {
        workload.push({
            chaincodeFunction: 'batch',
            chaincodeArguments: [type, start, amount],
        });
    } else {
        workload.push({
            'verb': 'batch',
            'type': type,
            'start': start,
            'amount': amount
        });
    }

    return workload;
}
```

변경 후(blockchain.bcType 으로 구분)

----------result----------

```
+------+--------+------+------+-----------+-------------+-------------+------------+------------+
| Test | Name   | Succ | Fail | Send Rate | Max Latency | Min Latency | Avg Latency | Throughput |
|------|--------|------|------|-----------|-------------|-------------|-------------|------------|
| 1    | mytest | 100  | 0    | 50.7 tps  | 1.23 s      | 0.50 s      | 0.83 s      | 39.2 tps   |
+------+--------+------+------+-----------+-------------+-------------+-------------+------------+

info: [monitor.js]:
 ### resource stats (maximum) ###
info: [monitor.js]:
+---------+-----------------------------------+-------------+---------+------------+-------------+-----------+------------+
| TYPE    | NAME                              | Memory(max) | CPU(max) | Traffic In | Traffic Out | Disc Read | Disc Write |
|---------|-----------------------------------|-------------|---------|------------|-------------|-----------|------------|
| Process | node local-client.js(avg)         | -           | NaN%    | -          | -           | -         | -          |
|---------|-----------------------------------|-------------|---------|------------|-------------|-----------|------------|
| Docker  | dev-peer0.org2.example.co...cc-v0 | 17.0MB      | 1.62%   | 60.6KB     | 11.6KB      | 24.0KB    | 0B         |
|---------|-----------------------------------|-------------|---------|------------|-------------|-----------|------------|
| Docker  | dev-peer1.org2.example.co...cc-v0 | 16.8MB      | 1.39%   | 59.8KB     | 11.4KB      | 24.0KB    | 0B         |
|---------|-----------------------------------|-------------|---------|------------|-------------|-----------|------------|
| Docker  | cli                               | 480.0KB     | 0.00%   | 0B         | 0B          | 0B        | 0B         |
|---------|-----------------------------------|-------------|---------|------------|-------------|-----------|------------|
| Docker  | kafka4.example.com                | 291.0MB     | 33.26%  | 336.9KB    | 1.3MB       | 0B        | 8.0KB      |
|---------|-----------------------------------|-------------|---------|------------|-------------|-----------|------------|
| Docker  | kafka3.example.com                | 279.1MB     | 6.88%   | 297.0KB    | 9.2KB       | 0B        | 8.0KB      |
|---------|-----------------------------------|-------------|---------|------------|-------------|-----------|------------|
| Docker  | peer0.org2.example.com            | 55.2MB      | 15.46%  | 277.8KB    | 127.6KB     | 0B        | 616.0KB    |
|---------|-----------------------------------|-------------|---------|------------|-------------|-----------|------------|
| Docker  | peer1.org2.example.com            | 32.6MB      | 14.32%  | 323.8KB    | 367.1KB     | 0B        | 616.0KB    |
|---------|-----------------------------------|-------------|---------|------------|-------------|-----------|------------|
| Docker  | zookeeper3.example.com            | 39.7MB      | 0.24%   | 1.0KB      | 820B        | 0B        | 0B         |
|---------|-----------------------------------|-------------|---------|------------|-------------|-----------|------------|
| Docker  | orderer3.example.com              | 17.1MB      | 4.68%   | 404.1KB    | 369.8KB     | 184.0KB   | 560.0KB    |
|---------|-----------------------------------|-------------|---------|------------|-------------|-----------|------------|
| Docker  | couchdb3                          | 89.5MB      | 6.77%   | 5.8KB      | 9.2KB       | 0B        | 24.0KB     |
|---------|-----------------------------------|-------------|---------|------------|-------------|-----------|------------|
| Docker  | couchdb2                          | 90.1MB      | 8.57%   | 5.9KB      | 9.1KB       | 0B        | 24.0KB     |
+---------+-----------------------------------+-------------+---------+------------+-------------+-----------+------------+
```

caliper 는 node2 에서 작동하기 때문에, report 에는 node2 에 존재하는 container 정보만 출력된다.
또한, benchmark config: command: end:를 수정했기 때문에, container 가 꺼지지 않고 유지된다.
node1 정보도 함께 출력하고자 한다면 benchmark config 파일을 수정해야한다.
보다 자세한 내용은 “Docker Daemon 설정”에서 다룬다.

5.4 Own Network & Own Chaincode & Distributed Client

```
---
test:
  name: my
  clients:
    type: local
    number: 1
  rounds:
    - label: mytest
```

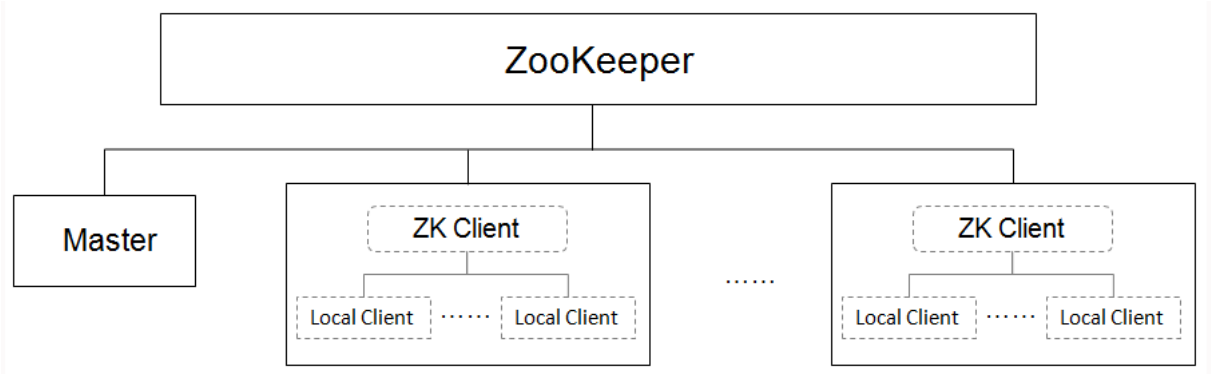
<local set>

```
---
test:
  name: simple
  clients:
    type: zookeeper
    zoo:
      server: localhost:2181
      clientsPerHost: 5
  rounds:
```

<zookeeper set>

(benchmark config 파일 中)

Caliper core 를 살펴보면, number: 의 수 만큼의 fork 를 한다. Caliper 는 각각의 fork 된 process 들에게 txn 을 균일하게 배분하고, 실행하게 한다. 뿐만 아니라 부하 분산을 위해 zookeeper 를 통한 테스트기능도 제공한다.



< Zookeeper Client Design >

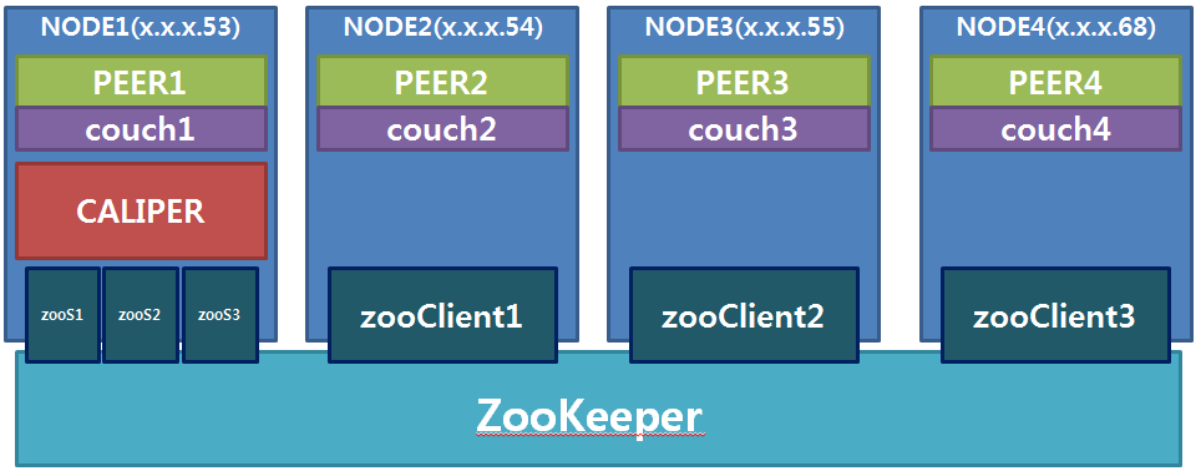
방법은 다음과 같다.

1) Fabric network 를 시작한다.
2) Caliper benchmark config 파일을 수정한다.
   2-1) zookeeper server 를 가동할 hostname:port 설정한다.
   2-2) zookeeper client 당 생성할 child client(clientPerHost 파라미터)수 설정한다.
3) Zookeeper server 를 시작한다..
4) 부하를 분산시킬 노드들로 zookeeper client 를 시작한다.
   단, zoo client 들도 마찬가지로 caliper 환경 구성을 해주어야 한다.

관련 파일 (run-benchmark.js 와 동일 경로)

- zookeeper-service.yaml
  zookeeper server 를 구성하는 컨테이너 정보
- zoo-service.js
  zookeeper server 를 시작하는 모듈
- start-zoo-client.js
  zookeeper client 를 시작하는 모듈

테스트 환경은 다음과 같다.



kafka 를 관리하는 zookeeper 는 따로 존재한다.(orderer node1, 2, 3 에 각각 하나씩)
위 그림에서 명시된 zookeeper 는 caliper client 를 관리하기 위한 zookeeper 이다.

---------Test flow----------

1) **Fabirc network 시작**

2) **Caliper benchmark config 파일을 수정한다.**

```
---
test:
  name: my
  clients:
    type: zookeeper
    zoo:
      server: 10.80.72.53:2181
      clientsPerHost: 5
  rounds:
  - label: mytest
    txNumber:
    - 100
    rateControl:
    - type: fixed-rate
      opts:
        tps: 50
    arguments:
      type: 'TKN'
      start: '1'
      amount: '100'
    callback: benchmark/simple/mymodule.js

monitor:
  type:
  - docker
  docker:
    name:
    - peer1.org1.komsco.com
    - http://10.80.72.54:1111/peer2.org1.komsco.com
    - http://10.80.72.55:1111/peer3.org1.komsco.com
    - http://10.80.72.68:1111/peer4.org1.komsco.com
    - http://10.80.72.57:1111/orderer1.komsco.com
    - http://10.80.72.58:1111/orderer2.komsco.com
    - http://10.80.72.59:1111/orderer3.komsco.com
    - couchdb1.peer1.org1.komsco.com
    - http://10.80.72.54:1111/couchdb2.peer2.org1.komsco.com
    - http://10.80.72.55:1111/couchdb3.peer3.org1.komsco.com
    - http://10.80.72.68:1111/couchdb4.peer4.org1.komsco.com
    - scripts_zoo1_1
    - scripts_zoo2_1
    - scripts_zoo3_1
    - prod-peer1.org1.komsco.com-channel1_setlcc-6
    - http://10.80.72.54:1111/prod-peer2.org1.komsco.com-channel1_setlcc-6
    - http://10.80.72.55:1111/prod-peer3.org1.komsco.com-channel1_setlcc-6
    - http://10.80.72.68:1111/prod-peer4.org1.komsco.com-channel1_setlcc-6

  interval: 1
```

Test 환경에서 zookeeper server 는 x.x.x.53 에서 작동하고,
zookeeper client 당 5 개의 child client 를 가진다.
그 외 다른 파라미터는 이전 테스트와 같고, 동일한 module 을 사용한다.
(monitor: docker: name: → caliper 에서 분리된 host 에 존재하는 containers 를
monitoring 하기 위함. "Docker Daemon 설정" 참고)

3) **Zookeeper server 를 시작한다.**

```
cd $CALIPER_HOME/packages/caliper-application/scripts
node zoo-service.js –t start
```

zoo-service.js module 은 zookeeper-service.yaml 에 입력한 container 들을 시작한다.
zookeeper-service.yaml 은 수정하지 않는다.

```
[root@jp-per01 scripts]# node zoo-service.js -t start
info: [scripts/zoo-service.js]: Starting zookeeper service ......
info: [scripts/zoo-service.js]: Executing command: docker-compose -f zookeeper-service.yaml up -d
Creating scripts_zoo3_1
Creating scripts_zoo1_1
Creating scripts_zoo2_1
info: [scripts/zoo-service.js]: Start zookeeper service successful
[root@jp-per01 scripts]# docker ps
CONTAINER ID        IMAGE
                                    NAMES
fb0f00872ffd        zookeeper:3.4.14
3888/tcp, 0.0.0.0:2182->2181/tcp   scripts_zoo2_1
49efe1dbc98f        zookeeper:3.4.14
0.0.0.0:2181->2181/tcp, 3888/tcp   scripts_zoo1_1
d9686f2bc573        zookeeper:3.4.14
3888/tcp, 0.0.0.0:2183->2181/tcp   scripts_zoo3_1
```

실행 결과이다.

4) **부하를 분산시킬 노드들로 zookeeper client 를 시작한다.**

이제, x.x.x.54, 55, 68 노드가 zookeeper server 에 붙으며 zookeeper client 역할을 해야한다.
54, 55, 68 노드들 또한 caliper 가 존재해야 하며, 동일한 fabric network 의

network config 파일과 module 이 존재해야 한다.

x.x.x.54

```
cd $CALIPER_HOME/packages/caliper-application/scripts
node start-zoo-client.js –n my_network_config –a “zookeeper_server_address:port”
```

x.x.x.55

```
cd $CALIPER_HOME/packages/caliper-application/scripts
node start-zoo-client.js –n my_network_config –a “zookeeper_server_address:port”
```

x.x.x.68

```
cd $CALIPER_HOME/packages/caliper-application/scripts
node start-zoo-client.js –n my_network_config –a “zookeeper_server_address:port”
```

- x.x.x.54 실행 결과

```
[root@jp-per02 scripts]# node start-zoo-client.js -n /home/peeradm/fabric-ccp-go-komsco.yaml -a "10.80.72.53:2181"
info: [scripts/zoo.js]: Starting zookeeper client of type fabric-ccp
info: [zoo-client.js]: Connected to ZooKeeper
info: [zoo-client.js]: Created client node:/caliper/clients/client_1559525101978_0000000000
info: [zoo-client.js]: Created receiving queue at:/caliper/client_1559525101978_0000000000_in
info: [zoo-client.js]: Created sending queue at:/caliper/client_1559525101978_0000000000_out
info: [zoo-client.js]: Waiting for messages at:/caliper/client_1559525101978_0000000000_in......
```

- x.x.x.55 실행 결과

```
[root@jp-per03 scripts]# node start-zoo-client.js -n /home/peeradm/fabric-ccp-go-komsco.yaml -a "10.80.72.53:2181"
info: [scripts/zoo.js]: Starting zookeeper client of type fabric-ccp
info: [zoo-client.js]: Connected to ZooKeeper
info: [zoo-client.js]: Created client node:/caliper/clients/client_1559525106510_0000000001
info: [zoo-client.js]: Created receiving queue at:/caliper/client_1559525106510_0000000001_in
info: [zoo-client.js]: Created sending queue at:/caliper/client_1559525106510_0000000001_out
info: [zoo-client.js]: Waiting for messages at:/caliper/client_1559525106510_0000000001_in......
```

- x.x.x.68 실행 결과

```
[root@jp-per05 scripts]# node start-zoo-client.js -n /home/peeradm/fabric-ccp-go-komsco.yaml -a "10.80.72.53:2181"
info: [scripts/zoo.js]: Starting zookeeper client of type fabric-ccp
info: [zoo-client.js]: Connected to ZooKeeper
info: [zoo-client.js]: Created client node:/caliper/clients/client_1559525100510_0000000002
info: [zoo-client.js]: Created receiving queue at:/caliper/client_1559525100510_0000000002_in
info: [zoo-client.js]: Created sending queue at:/caliper/client_1559525100510_0000000002_out
info: [zoo-client.js]: Waiting for messages at:/caliper/client_1559525100510_0000000002_in......
```

5) Caliper 시작

이제, caliper 를 가동하면, zookeeper server(x.x.x.53)에서 zookeeper client(54, 55, 68)들에게 task 를 분배하여 할당한다. { txNumber / clients 만큼(현 테스트는 100 / 3 = 약 33 개) } zookeeper client 는각자 clientsPerHost 만큼의 fork 를 통해 이를 병렬수행한다.
즉, 하나의 process 마다 33/5 개 만큼 txn 을 발생시키는 셈이다.

- caliper core 실행 결과

```
info: [caliper-flow]:
+------+--------+------+------+-----------+-------------+-------------+-------------+------------+
| Test | Name   | Succ | Fail | Send Rate | Max Latency | Min Latency | Avg Latency | Throughput |
|------|--------|------|------|-----------|-------------|-------------|-------------|------------|
| 1    | mytest | 99   | 0    | 10.7 tps  | 1.33 s      | 0.13 s      | 0.83 s      | 9.9 tps    |
+------+--------+------+------+-----------+-------------+-------------+-------------+------------+

info: [monitor.js]:
 ### resource stats (maximum) ###
info: [monitor.js]:
+--------+-----------------------------------+-------------+----------+-------------+--------------+-----------+------------+
| TYPE   | NAME                              | Memory(max) | CPU(max) | Traffic In  | Traffic Out  | Disc Read | Disc Write |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | 10.80.72.59/orderer3.koms...o.com | 566.2MB     | 4.06%    | 412.2KB     | 478.1KB      | 0B        | 180.0KB    |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | 10.80.72.58/orderer2.koms...o.com | 570.1MB     | 4.82%    | 280.9KB     | 352.6KB      | 0B        | 288.0KB    |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | 10.80.72.57/orderer1.koms...o.com | 570.8MB     | 2.54%    | 293.0KB     | 353.8KB      | 0B        | 288.0KB    |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | 10.80.72.54/prod-peer2.or...lcc-6 | 29.0MB      | 0.00%    | 157B        | 149B         | 0B        | 0B         |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | 10.80.72.54/peer2.org1.ko...o.com | 1.1GB       | 24.15%   | 300.7KB     | 14.8MB       | 0B        | 208.0KB    |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | 10.80.72.54/couchdb2.peer...o.com | 1.0GB       | 4.96%    | 1.4KB       | 2.0KB        | 0B        | 0B         |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | 10.80.72.68/prod-peer4.or...lcc-6 | 25.1MB      | 0.00%    | 74B         | 0B           | 0B        | 0B         |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | 10.80.72.68/peer4.org1.ko...o.com | 857.0MB     | 9.30%    | 207.1KB     | 115.9KB      | 0B        | 208.0KB    |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | 10.80.72.68/couchdb4.peer...o.com | 1.1GB       | 4.73%    | 1.4KB       | 2.0KB        | 0B        | 0B         |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | scripts_zoo2_1                    | 82.6MB      | 1.37%    | 4.9KB       | 1.8KB        | 0B        | 44.0KB     |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | scripts_zoo1_1                    | 84.5MB      | 1.79%    | 11.4KB      | 12.9KB       | 0B        | 48.0KB     |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | scripts_zoo3_1                    | 98.6MB      | 1.29%    | 7.1KB       | 10.0KB       | 0B        | 44.0KB     |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | prod-peer1.org1.komsco.co...lcc-6 | 32.7MB      | 0.00%    | 74B         | 0B           | 0B        | 0B         |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | peer1.org1.komsco.com             | 1.4GB       | 42.49%   | 390.1KB     | 26.9MB       | 0B        | 336.0KB    |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | couchdb1.peer1.org1.komsco.com    | 1.1GB       | 3.79%    | 2.6KB       | 3.8KB        | 0B        | 88.0KB     |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | 10.80.72.55/prod-peer3.or...lcc-6 | 32.2MB      | 0.00%    | 74B         | 0B           | 0B        | 0B         |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | 10.80.72.55/peer3.org1.ko...o.com | 852.5MB     | 9.59%    | 295.0KB     | 129.4KB      | 0B        | 340.0KB    |
|--------|-----------------------------------|-------------|----------|-------------|--------------|-----------|------------|
| Docker | 10.80.72.55/couchdb3.peer...o.com | 1.1GB       | 2.09%    | 2.6KB       | 3.8KB        | 0B        | 88.0KB     |
+--------+-----------------------------------+-------------+----------+-------------+--------------+-----------+------------+
```

- x.x.x.54, 55 실행결과 (68 은 생략)

```
[root@jp-per02 scripts]# node start-zoo-client.js -n /home/peeradm/fabric-ccp-go-komsco.yaml -a "10.80.72.53:2181"
info: [scripts/zoo.js]: Starting zookeeper client of type fabric-ccp
info: [zoo-client.js]: Connected to ZooKeeper
info: [zoo-client.js]: Created client node:/caliper/clients/client_1559527667783_0000000012
info: [zoo-client.js]: Created receiving queue at:/caliper/client_1559527667783_0000000012_in
info: [zoo-client.js]: Created sending queue at:/caliper/client_1559527667783_0000000012_out
info: [zoo-client.js]: Waiting for messages at:/caliper/client_1559527667783_0000000012_in......
info: [zoo-client.js]: Receive message, type=test
info: [client-util.js]: All clients ready, starting test phase
info: [local-client.js]: txUpdateTime: 1000
info: [local-client.js]: txUpdateTime: 1000
info: [local-client.js]: txUpdateTime: 1000
info: [local-client.js]: txUpdateTime: 1000
info: [local-client.js]: txUpdateTime: 1000
info: [local-client.js]: Info: client 31554 start test runFixedNumber()
info: [local-client.js]: Info: client 31537 start test runFixedNumber()
info: [local-client.js]: Info: client 31542 start test runFixedNumber()
info: [local-client.js]: Info: client 31549 start test runFixedNumber()
info: [local-client.js]: Info: client 31543 start test runFixedNumber()
info: [zoo-client.js]: Receive message, type=quit
info: [client-util.js]: Client exited
info: [client-util.js]: Client exited
info: [client-util.js]: Client exited
info: [client-util.js]: Client exited
info: [client-util.js]: Client exited
```

```
[root@jp-per03 scripts]# node start-zoo-client.js -n /home/peeradm/fabric-ccp-go-komsco.yaml -a "10.80.72.53:2181"
info: [scripts/zoo.js]: Starting zookeeper client of type fabric-ccp
info: [zoo-client.js]: Connected to ZooKeeper
info: [zoo-client.js]: Created client node:/caliper/clients/client_1559527669661_0000000013
info: [zoo-client.js]: Created receiving queue at:/caliper/client_1559527669661_0000000013_in
info: [zoo-client.js]: Created sending queue at:/caliper/client_1559527669661_0000000013_out
info: [zoo-client.js]: Waiting for messages at:/caliper/client_1559527669661_0000000013_in......
info: [zoo-client.js]: Receive message, type=test
info: [client-util.js]: All clients ready, starting test phase
info: [local-client.js]: txUpdateTime: 1000
info: [local-client.js]: txUpdateTime: 1000
info: [local-client.js]: txUpdateTime: 1000
info: [local-client.js]: txUpdateTime: 1000
info: [local-client.js]: txUpdateTime: 1000
info: [local-client.js]: Info: client 6616 start test runFixedNumber()
info: [local-client.js]: Info: client 6610 start test runFixedNumber()
info: [local-client.js]: Info: client 6632 start test runFixedNumber()
info: [local-client.js]: Info: client 6626 start test runFixedNumber()
info: [local-client.js]: Info: client 6615 start test runFixedNumber()
info: [zoo-client.js]: Receive message, type=quit
info: [client-util.js]: Client exited
info: [client-util.js]: Client exited
info: [client-util.js]: Client exited
info: [client-util.js]: Client exited
info: [client-util.js]: Client exited
```

6. Docker daemon 설정

```
monitor:
  type:
  - docker
  docker:
    name:
    - peer1.org1.komsco.com
    - http://10.80.72.54:1111/peer2.org1.komsco.com
    - http://10.80.72.55:1111/peer3.org1.komsco.com
    - http://10.80.72.68:1111/peer4.org1.komsco.com
    - http://10.80.72.57:1111/orderer1.komsco.com
    - http://10.80.72.58:1111/orderer2.komsco.com
    - http://10.80.72.59:1111/orderer3.komsco.com
    - couchdb1.peer1.org1.komsco.com
    - http://10.80.72.54:1111/couchdb2.peer2.org1.komsco.com
    - http://10.80.72.55:1111/couchdb3.peer3.org1.komsco.com
    - http://10.80.72.68:1111/couchdb4.peer4.org1.komsco.com
    - scripts_zoo1_1
    - scripts_zoo2_1
    - scripts_zoo3_1
    - prod-peer1.org1.komsco.com-channel1_setlcc-6
    - http://10.80.72.54:1111/prod-peer2.org1.komsco.com-channel1_setlcc-6
    - http://10.80.72.55:1111/prod-peer3.org1.komsco.com-channel1_setlcc-6
    - http://10.80.72.68:1111/prod-peer4.org1.komsco.com-channel1_setlcc-6

  interval: 1
```

-- benchmark config 파일 中

caliper 에서는 다른 노드의 container 의 모니터링을 위해 다음의 설정을 해주어야 한다.
(http://ADDRESS:PORT/CONTAINER_NAME)

1) HTTP REST 형식으로 정보를 받기 위해 Docker daemon 의 소켓설정을 수정한다.

```
vim /usr/lib/systemd/system/docker.service
```

```
[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H unix:// -H tcp://0.0.0.0:1111
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always
```

Docker 는 daemon 과 client 간 통신에서 local 은 unix socket 을, 원격에서는 tcp socket 을 사용한다. 우리는 분리된 노드 간 container 의 monitoring 이 필요하기에 tcp socket 을 추가한다.
(-H tcp://0.0.0.0:port 추가) default port: 4243
수정 후 다음을 통해 remote API 를 가능케 한다.

2) Reload the docker daemon

```
systemctl daemon-reload
```

3) Restart the container (container 가 작동 중일 시)

```
sudo service docker restart
```

4) Test

```
curl http://host:port/images/json
```

다른 노드에서 curl 을 통해 데이터를 가져올 수 있는지 확인한다.

7. MVCC CONFILCT 해결

어떠한 테스트를 하느냐에 따라 다르겠지만, 테스트를 진행하다 보면 MVCC confilct 가 나는 상황을 종종 발견할 수 있다. 한 블록에 동일 데이터를 수정하는 txn 이 존재한다면 MVCC 를 피할 수 없을것이다. ex) 송금관련 chaincode 테스트
본 챕터에서 MVCC 를 피해가기 위해 시도했던 3 가지 방법을 서술한다.

7.1 Module 수정 1 (Loop)

benchmark config 내 callback: 파라미터는 caliper test 에서 사용할 module 을 명시하도록 되어있다.

```yaml
test:
  name: simple
  description: This is an example benchr
    performance with simple account oper
  clients:
    type: local
    number: 3
  rounds:
  - label: open
    description: Test description for th
    txNumber:
    - 100
    rateControl:
    - type: fixed-rate
      opts:
        tps: 50
    arguments:
      money: 1000
      txnPerBatch: 10
    callback: benchmark/simple/open.js
```

→ config.yaml (benchmark config file)

그리고 가르키는 module 을 열어보면,

```javascript
function generateWorkload() {
    let workload = [];
    for(let i= 0; i < txnPerBatch; i++) {
        let acc_id = generateAccount();
        account_array.push(acc_id);

        if (bc.bcType === 'fabric-ccp') {
            workload.push({
                chaincodeFunction: 'open',
                chaincodeArguments: [acc_id, initMoney.toString()],
            });
        } else {
            workload.push({
                'verb': 'open',
                'account': acc_id,
                'money': initMoney
            });
        }
    }
    return workload;
}
```

→ open.js(caliper 제공 sample)

```javascript
module.exports.run = function() {
    let args = generateWorkload();
    return bc.invokeSmartContract(contx, 'simple', 'v0', args, 100);
};
```

→ open.js

benchmark config 파일에서 받은 arguments 를 변수에 담고, array 에 추가하여 최종적으로 bc.invokeSmartContract()의 인자로 넘겨주고 있다.
MVCC 는 arguments 의 중복으로 발생한다고도 볼 수 있기 때문에, arguments 를 loop 을 통해 바꾸어 넘겨주는 방법을 생각해 볼 수 있다.
sample module(open.js)에서는 txnPerBatch 라는 파라미터를 benchmark config 에서 받아 해당 숫자 만큼 loop 하고 있다. 이 loop 문을 이용해 array 에 변하는 arguments 를 적재하는 방법이다.

하지만, caliper 에서는 “It is recommended that only one transaction is submitted in each all, but this is not a MUST requirement. If multiple transactions are submitted each time, the

actual workload may be different with the configured workload." 라고 말하고 있다.
따라서, 이 방법은 간단하지만 caliper 에서 권장하는 one call, one txn 이 아니다.
실제 수행해본 결과 원활한 테스트가 불가능 하였다. (향후 개선될 것으로 예상 되어짐)

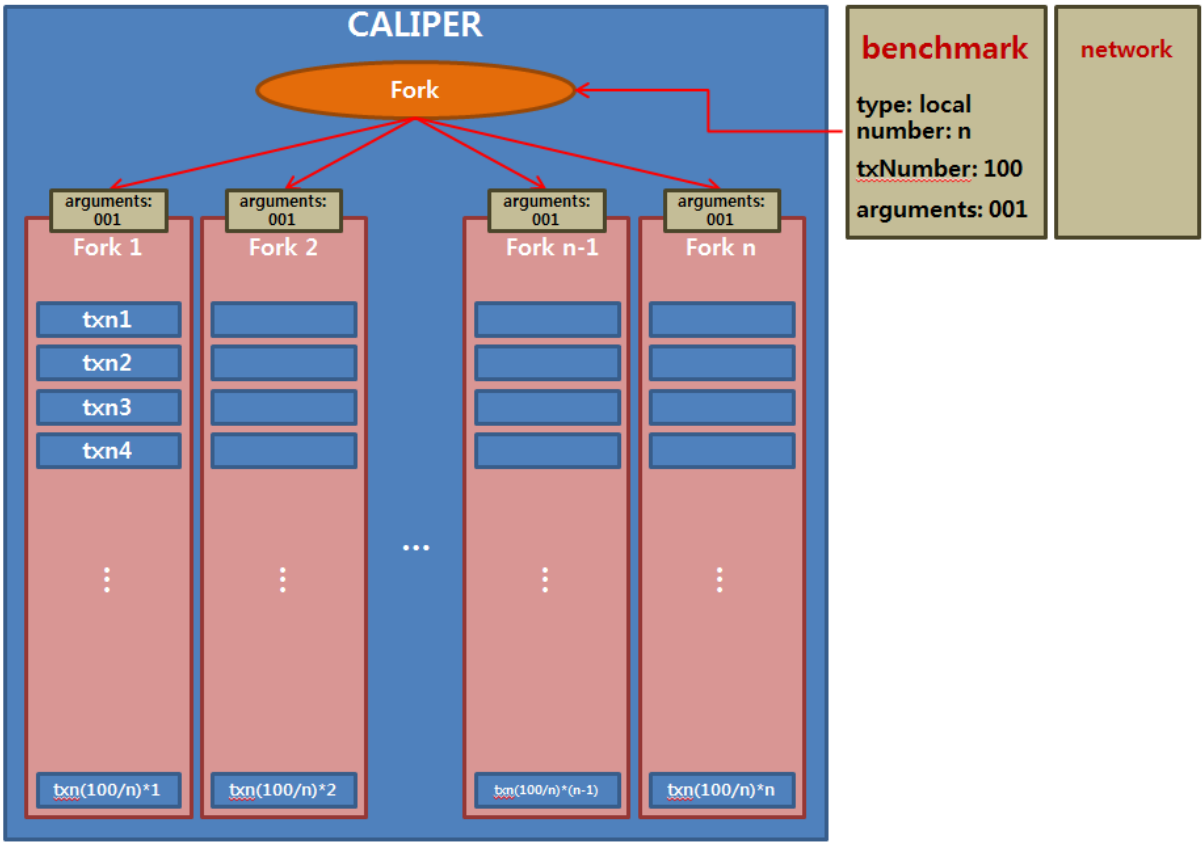
7.2 Module 수정 2 (Random)

One call, one txn 을 위해 Loop 을 없애고, 매 call 마다 랜덤한 값을 생성하여 중복을 피하는 방법이다.
one call, one txn 은 만족하지만, 확률상으로 100% MVCC 를 피하기는 힘들다.

7.3 Caliper core 수정

중복된 값을 피하고, one call, one txn 을 가능케 하는 방법이다. Core 단 수정이기 때문에 clients type(local, zookeeper)에 따라 설정을 달리 해야 한다.

7.3.1 Local



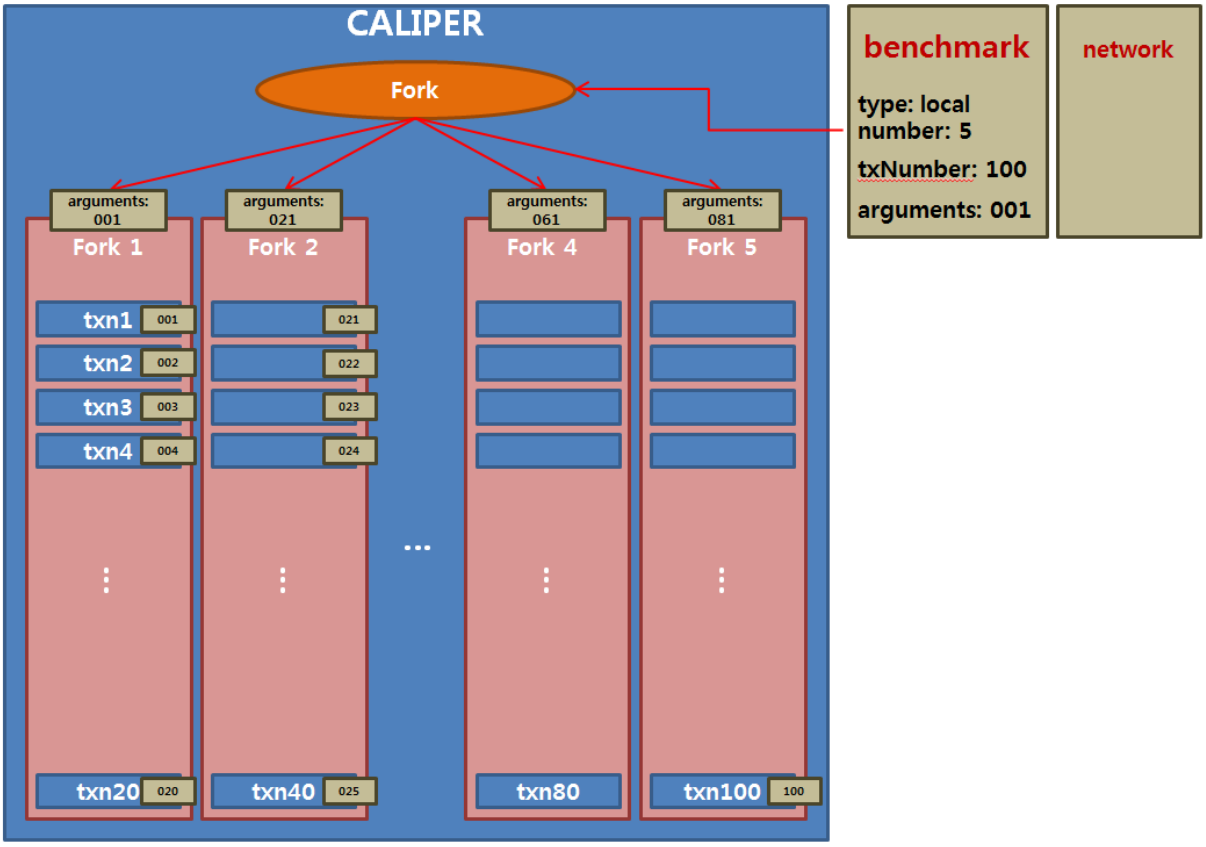
caliper flow 를 구조화 한 모습이다.

1) benchmark config 에서 number: 만큼 fork 한다.
2) (총 txn 의 수 / fork 수)만큼 각 fork 에게 할당한다. (txNumber / number)
   동시에 benchmark config 의 arguments 값을 넘겨준다.
3) 각 fork 들은 arguments 를 토대로 txn 을 발생시킨다

위 과정에서 2) 3)을 수정하면 MVCC 를 피할 수 있다.

1) 동일
2) (총 txn 의 수 / fork 수)만큼 각 fork 에게 할당한다. (txNumber / number)
   benchmark config 에서의 arguments 를 받아 겹치지 않는 값으로 변경하고 넘긴다.
   각 fork 들은 다른 arguments 를 전달받는다.

3) 각 fork 들은 arguments 를 토대로 txn 를 발생시킨다.
한번 발생 시킨 후 arguments 를 변경한다.



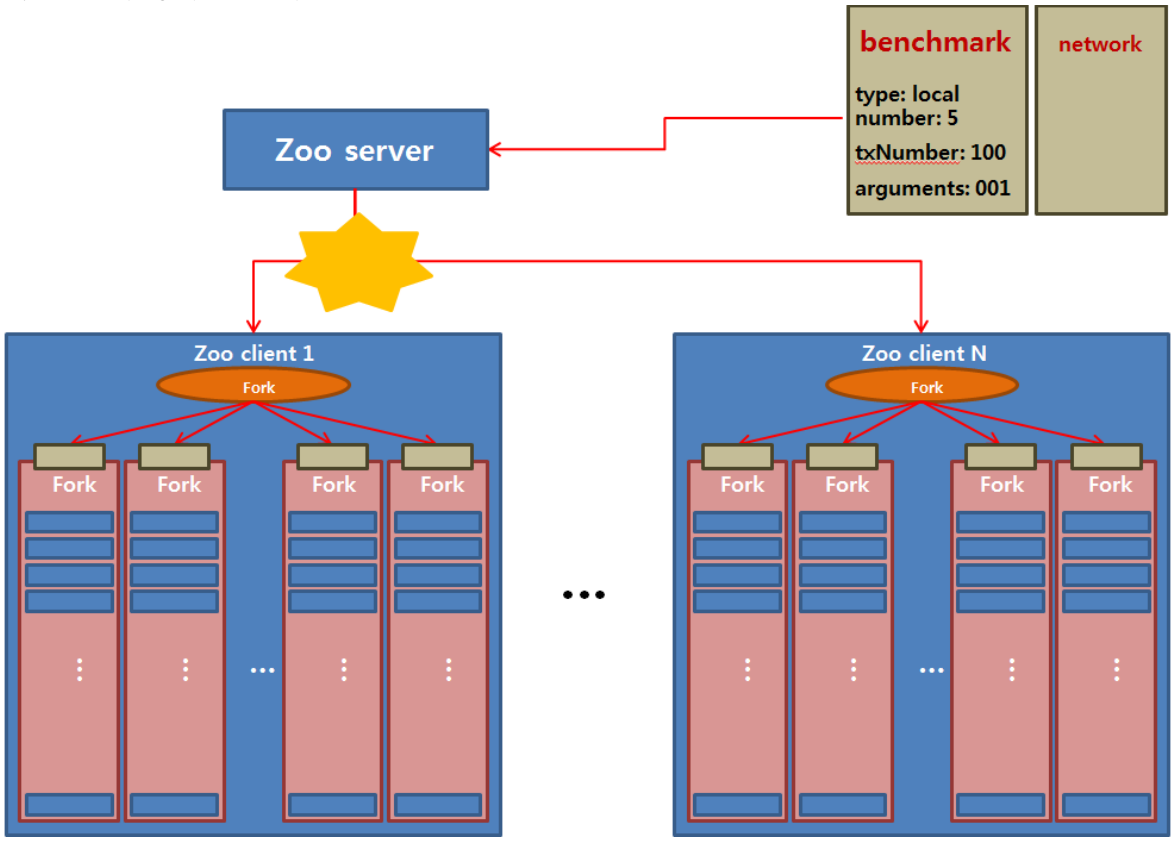
→ Modified caliper core

예시

다음은 각 fork 들이 서로 다른 arguments 를 전달받고,
한번의 txn 발생 후 arguments 값을 1 씩 증가시키는 상황이다.
(arguments: 지갑번호 txn: 송금)이라고 가정한다면, MVCC 를 배제한 상태로 테스트할 수 있다.

7.3.2 Zookeeper

Client type: zookeeper 또한 어렵지않다.
7.3.1 의 로직은 그대로 유지한 체, zoo client 에게 할당하는 arguments 를 비슷한 방법으로 수정하면 된다.



위 그림의 노란색 별 모양에서 한번의 수정을 추가한다.

- $CALIPER_HOME/packages/node_modules/caliper-core/lib/caliper-flow.js
  : caliper 전체 flow
- $CALIPER_HOME/packages/node_modules/caliper-core/lib/client/*
  : 7.3 을 위해 수정해야 할 모듈들