

연구 보고서

작성자	김한호	작성일자	2021.05.22.
1. 연구 계획			
<ul style="list-style-type: none"> - 체인코드 함수 작성 테스트코드 작성 - 논문 검색 후 타 논문 연구 검색지속 			
2. 논문 연구 진행			
<pre> 159 func TestVerifyIsPermitted(t *testing.T) { 160 transactionContext, chaincodeStub := prepMocksAsOrg1() 161 civilTransferCC := chaincode.SmartContract{} 162 testCivilTransientInput := &civilTransientInput{ 163 ID: "civil1", 164 Name: "John", 165 PhoneNumber: "010-1234-3213", 166 Address: "Wonchen-dong, Suwon-si", 167 Status: "Normal", 168 } 169 setReturnCivilPropsInTransientMap(t, chaincodeStub, testCivilTransientInput) 170 //fmt.Printf("testCivil: %v\n", testCivil) 171 err := civilTransferCC.CreateCivil(transactionContext) 172 require.NoError(t, err) 173 calledCollection, calledId, calledCivilBytes := chaincodeStub.PutPrivateDataArgsForCall(0 174) 175 //calledBytesAsJSON, err := json.Marshal(calledCivilBytes) 176 fmt.Printf("calledCollection: %v calledId: %v calledCivilBytes: %v\n", calledCollection, 177 calledId, string(calledCivilBytes)) 178 //testCivil := new(Civil) 179 //json.Unmarshal(calledCivilBytes, testCivil) 180 fmt.Printf("unmarshal Civil: %v\n", string(calledCivilBytes.ID)) 181 testReqPrivacy := &chaincode.ReqPrivacy{ 182 ID: "id1", 183 IsPermitted: true, 184 } 185 reqprivBytes, err := json.Marshal(testReqPrivacy) 186 if err != nil { 187 fmt.Errorf("json marshal something wrong") 188 } 189 setReturnReqPrivDataInStub(t, chaincodeStub, testReqPrivacy) 190 fmt.Printf("reqprivBytes: %v", reqprivBytes) 191 } </pre>			
<p>조직에서 개인정보를 요청하면 시민과 조직 개인정보 허가가 되어 있는지 확인하기 위한 로직을 테스트하는 함수를 작성함. asset-transfer-private-data예제에서는 동일한 구조의 변수를 갖고 있기 때문에 해시(Hash)값을 구해서 비교 후 동일하면 에러 값을 반환하지 않은 상태로 진행함. 프로젝트 설계에서 동일한 속성값을 갖추고 있었지만 구조체(struct)의 갖고 있는 속성이 다르기 때문에 기존 논리를 통해 프로젝트를 진행할 수 없음을 알게 됨.</p>			

```

16 func TestReadCivil(t *testing.T) {
17     chaincodeStub := &mocks.ChaincodeStub{}
18     transactionContext := &mocks.TransactionContext{}
19     transactionContext.GetStubReturns(chaincodeStub)
20
21     expectedCivil := &chaincode.Civil{ID: "id1"}
22     bytes, err := json.Marshal(expectedCivil)
23     require.NoError(t, err)
24
25     chaincodeStub.GetStateReturns(bytes, nil)
26     e2listQueriesCC := chaincode.SmartContract{}
27     civil, err := e2listQueriesCC.ReadCivil(transactionContext, "")
28     require.NoError(t, err)
29     require.Equal(t, expectedCivil, civil)
30
31     chaincodeStub.GetStateReturns(nil, fmt.Errorf("unable to retrieve civil"))
32     _, err = e2listQueriesCC.ReadCivil(transactionContext, "")
33     require.EqualError(t, err, "failed to read from world state: unable to retrieve civil")
34
35     chaincodeStub.GetStateReturns(nil, nil)
36     civil, err = e2listQueriesCC.ReadCivil(transactionContext, "id1")
37     require.EqualError(t, err, "the civil id1 does not exists")
38     require.Nil(t, civil)
39 }
40
41 func TestReadReqPrivacy(t *testing.T) {
42     transactionContext, chaincodeStub := prepMocksAsOrg1()
43     e2listQueriesCC := chaincode.SmartContract{}

```

시민(Civil)을 읽어오는 함수를 테스트함에 있어서도 기존에 private-data에서는 직접 함수를 작성해서 GetPrivateDataReturns 함수를 불러와서 테스트를 함. asset-transfer-basic에서는 GetStateReturns를 이용해 테스트를 함.

```

1 package chaincode
2
3 import (
4     "encoding/json"
5     "fmt"
6     "log"
7
8     "github.com/hyperledger/fabric-contract-api-go/contractapi"
9 )
10
11 // ReadReqPrivacy read the information from collection
12 func (s *SmartContract) ReadReqPrivacy(ctx contractapi.TransactionContextInterface, reqprivID string) (*ReqPrivacy, error) {
13     log.Printf("ReadReqPrivacy: collection %v, ID %v", civilCollection, reqprivID)
14     reqprivJSON, err := ctx.GetStub().GetPrivacyData(civilCollection, reqprivID) //get the reqprivacy from chaincode state
15     if err != nil {
16         return nil, fmt.Errorf("failed to read reqprivacy: %v", err)
17     }
18
19     // No ReqPrivacy found, return empty response
20     if reqprivJSON == nil {
21         log.Printf("%v does not exist in collection %v", reqprivID, civilCollection)
22         return nil, nil
23     }
24
25     var reqpriv *ReqPrivacy
26     err = json.Unmarshal(reqprivJSON, &reqpriv)
27     if err != nil {
28         return nil, fmt.Errorf("failed to unmarshal JSON: %v", err)
29     }
30     return reqpriv, nil
31 }

```

읽어오는 함수를 작성하여 차후 값을 읽어오는 과정에서 다시 활용할 수 있도록 구현함.

```

Civil already exists: id1
2021/05/22 16:11:31 CreateCivil Put: collection: civilCollection, ID id1, client 0#00U,z00
2021/05/22 16:11:31 Civil Private Details Put: collection Org1TestmspPrivateCollection, ID id1
2021/05/22 16:11:31 CreateCivil InfectionStatus: collection: civilCollection, ID id1, client 0#00U,z00
2021/05/22 16:11:31 CreateCivil Put: collection: civilCollection, ID civil1, client 0#00U,z00
2021/05/22 16:11:31 Civil Private Details Put: collection Org1TestmspPrivateCollection, ID civil1
2021/05/22 16:11:31 CreateCivil InfectionStatus: collection: civilCollection, ID civil1, client 0#00U,z00
calledCollection: civilCollection calledId: civil1 calledCivilBytes: {"civilID":"civil1"}
reqprivBytes: [123 34 114 101 113 112 114 105 118 73 68 34 58 34 105 100 49 34 44 34 99 105 118 1
05 108 73 68 34 58 34 34 44 34 99 111 110 116 114 111 108 73 68 34 58 34 34 44 34 114 101 113 115
116 97 114 116 68 97 116 101 34 58 34 34 44 34 114 101 113 101 110 100 68 97 116 101 34 58 34 34
44 34 105 115 80 101 114 109 105 116 116 101 100 34 58 116 114 117 101 125]2021/05/22 16:11:31 R
eadReqPrivacy: collection controlCollection, ID id1
2021/05/22 16:11:31 id1 does not exist in collection controlCollection
2021/05/22 16:11:31 ReadReqPrivacy: collection controlCollection, ID id1
2021/05/22 16:11:31 ReadReqPrivacy: collection controlCollection, ID id1
2021/05/22 16:11:31 ReadResPrivacy: collection civilCollection, ID id1
2021/05/22 16:11:31 id1 does not exist in collection civilCollection
2021/05/22 16:11:31 ReadResPrivacy: collection civilCollection, ID id1
2021/05/22 16:11:31 ReadResPrivacy: collection civilCollection, ID id1
resprivRead :&{id1 true} testResPrivacy: &{id1 true}PASS
ok      github.com/hyperledger/fabric-samples/asset-transfer-private-data/chaincode-go/chaincode0
.035s

```

읽어오는 함수를 작성하고 go test명령어를 실행함. 화면을 통해 값을 읽어오는 과정과 중간에 출력을 하는 부분을 확인할 수 있음.

```

// AgreeToPrivacy is used by the request control of the privacy to agree to the
// civil. The agree to IsPermitted value is stored in the civil orgs
// org specific collection, while the the civil client ID is sotred in the civil collection
// using a composite key
func (s *SmartContract) AgreeToPrivacy(ctx contractapi.TransactionInterface) error {
    // Get ID of submitting client identity
    clientID, err := ctx.GetClientIdentity().GetID()
    if err != nil {
        return fmt.Errorf("failed to get verified OrgID: %v", err)
    }
    // Value is private, therefore it gets passed in transient field
    transientMap, err := ctx.GetStub().GetTransient()
    if err != nil {
        return fmt.Errorf("error getting transient: %v", err)
    }
    // Persist the JSON bytes as-is so that there is no risk of nondeterministic marshaling
    valueJSONAsBytes, ok := transientMap["privacy_properties"]
    if !ok {
        return fmt.Errorf("privacy_value key not found in the transient map")
    }
    // Unmarshal the transient map to get the respivID
    var valueJSON ResPrivacy
    err = json.Unmarshal(valueJSONAsBytes, &valueJSON)
    if err != nil {
        return fmt.Errorf("failed to unmarshal JSON: %v", err)
    }

    // Do some error checking since logical wrong
    if len(valueJSON.ID) == 0 {
        return fmt.Errorf("resprivID field must be a non-empty string")
    }
    if valueJSON.IsPermitted == false {
        return fmt.Errorf("isPermittedValue field must be a true")
    }

    // Read ReqPrivacy from the private data collection
    reqprivacy, err := s.ReadReqPrivacy(ctx, valueJSON.ID)
}

```

합의하는 과정이 들어가는 함수를 작성하기 시작함. 실행하는 조직을 확인이 안 될 경우 에러를 발생함. 읽은 값을 저장하여 블록에 저장 후에 읽음. 값을 통해

서로 합의가 되었는지 확인을 함.


```
2021/05/18 13:57:43 CreateCivil Put: collection: civilCollection, ID id1, client 0#00U,z00
2021/05/18 13:57:43 Put: collection Org1TestmspPrivateCollection, ID id1
2021/05/18 13:57:43 CreateCivil InfectionStatus: collection: civilCollection, ID id1, client 0#00U,z00
2021/05/18 13:57:43 ReadReqPrivacy: collection controlCollection, ID id1
2021/05/18 13:57:43 id1 does not exist in collection controlCollection
2021/05/18 13:57:43 ReadReqPrivacy: collection controlCollection, ID id1
2021/05/18 13:57:43 ReadReqPrivacy: collection controlCollection, ID id1
2021/05/18 13:57:43 ReadResPrivacy: collection civilCollection, ID id1
2021/05/18 13:57:43 id1 does not exist in collection civilCollection
2021/05/18 13:57:43 ReadResPrivacy: collection civilCollection, ID id1
2021/05/18 13:57:43 ReadResPrivacy: collection civilCollection, ID id1
PASS
ok      github.com/hyperledger/fabric-samples/asset-transfer-private-data/chaincode-go/chaincode0
.041s
h2kim@h2kim-VBox:~/fabric-samples/e2-list-private-data/chaincode-go/chaincode$ ls
asset_queries.go      asset_transfer_test.go  e2_list_queries.go
asset_queries_test.go  e2_list_enter.go        e2_list_queries_test.go
asset_transfer.go      e2_list_enter_test.go   mocks
h2kim@h2kim-VBox:~/fabric-samples/e2-list-private-data/chaincode-go/chaincode$ cd ..
h2kim@h2kim-VBox:~/fabric-samples/e2-list-private-data/chaincode-go$ ls
META-INF  README.md  chaincode  collections_config.json  go.mod  go.sum  main.go  vendor
h2kim@h2kim-VBox:~/fabric-samples/e2-list-private-data/chaincode-go$ cd ..
h2kim@h2kim-VBox:~/fabric-samples/e2-list-private-data$ ls
README.md  application-javascript  chaincode-go
h2kim@h2kim-VBox:~/fabric-samples/e2-list-private-data$ git add .
h2kim@h2kim-VBox:~/fabric-samples/e2-list-private-data$ git commit -m "add ReadReqPrivacy, ReadResPrivacy"
[main a1b1acc] add ReadReqPrivacy, ReadResPrivacy
5 files changed, 74 insertions(+), 7 deletions(-)
h2kim@h2kim-VBox:~/fabric-samples/e2-list-private-data$ git push
Username for 'https://github.com': ik2ki
Password for 'https://ik2ki@github.com':
오브젝트 개수 세는 중: 9, 완료.
오브젝트 압축하는 중: 100% (9/9), 완료.
오브젝트 쓰는 중: 100% (9/9), 1.14 KiB | 1.14 MiB/s, 완료.
Total 9 (delta 8), reused 0 (delta 0)
remote: Resolving deltas: 100% (8/8), completed with 8 local objects.
To https://github.com/ik2ki/e2-list-private-data.git
```

테스트 이후 구현을 했던 코드를 git에 올리는 과정을 수행함.

```
h2kim@h2kim-VBox:~/fabric-samples$ cp -r asset-transfer-basic/ e2-enter-basic
h2kim@h2kim-VBox:~/fabric-samples$ cd e2-enter-basic/
h2kim@h2kim-VBox:~/fabric-samples/e2-enter-basic$ ls
application-go  application-javascript  chaincode-go  chaincode-javascript
application-java  chaincode-external  chaincode-java  chaincode-typescript
h2kim@h2kim-VBox:~/fabric-samples/e2-enter-basic$ find . ! -path "./*.chaincode-go*" | cut -d "." -f2 | cut -d "/" -f2 | xargs rm -rf {} \;
h2kim@h2kim-VBox:~/fabric-samples/e2-enter-basic$ ls
chaincode-go
h2kim@h2kim-VBox:~/fabric-samples/e2-enter-basic$
```

논문을 찾아본 이후 진행을 하는 private-data를 구현하여 완성을 하는 것 보다 여러 값을 주어서 실험 결과값을 도출하는 과정을 먼저 하는 것이 중요하다고 판단함. 지금 private-data부분은 진행하는 것은 설정의 문제와 시간이 걸릴 것으로 예측이 되고 여러 실험값을 변형하면서 caliper-benchmark의 다양한 결과 값을 다 구하는 것이 유용하지 않다고 판단이 됨. asset-transfer-basic을 통해 여러 논문을 통해 생각했던 다양한 환경에서 결과 값을 낸 후. 중간 저장되는 값을 공개 키 암호화 방식을 적용하고 위에 적용했던 실험을 동일 반복 후 결과를 낸 이후 마지막으로 private-data를 적용한 실험 값을 적용하여 결론을 내는 것이 나을 것이라 판단함.

Quick setup — if you've done this kind of thing before

 Set up in Desktop

or

HTTPS

SSH

<https://github.com/ik2ki/e2-enter-basic.git>



Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# e2-enter-basic" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/ik2ki/e2-enter-basic.git
git push -u origin main
```



...or push an existing repository from the command line

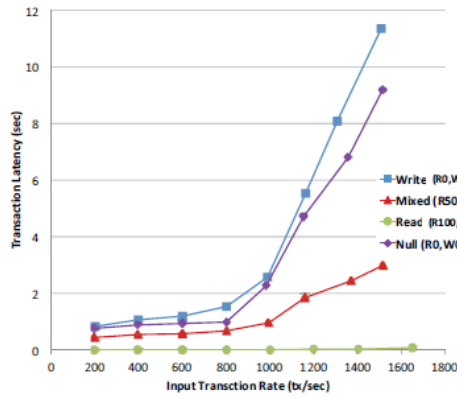
```
git remote add origin https://github.com/ik2ki/e2-enter-basic.git
git branch -M main
git push -u origin main
```



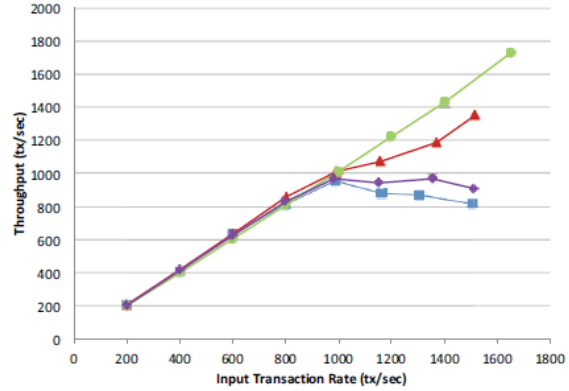
새롭게 e2-enter-basic git을 생성하여 위에 과정을 따라하기로 함.

```
h2kim@h2kim-VBox:~/fabric-samples/e2-enter-basic$ git branch -M main
h2kim@h2kim-VBox:~/fabric-samples/e2-enter-basic$ git remote add origin https://github.com/ik2ki/
e2-enter-basic.git
h2kim@h2kim-VBox:~/fabric-samples/e2-enter-basic$ git push -u origin main
Username for 'https://github.com': ik2ki
Password for 'https://ik2ki@github.com':
오브젝트 개수 세는 중: 888, 완료.
오브젝트 압축하는 중: 100% (825/825), 완료.
오브젝트 쓰는 중: 100% (888/888), 2.27 MiB | 2.10 MiB/s, 완료.
Total 888 (delta 190), reused 0 (delta 0)
remote: Resolving deltas: 100% (190/190), done.
To https://github.com/ik2ki/e2-enter-basic.git
 * [new branch]      main -> main
'main' 브랜치가 리모트의 'main' 브랜치를 ('origin'에서) 따라가도록 설정되었습니다.
h2kim@h2kim-VBox:~/fabric-samples/e2-enter-basic$
```

위에 과정을 실행한 결과 모습을 볼 수 있음.



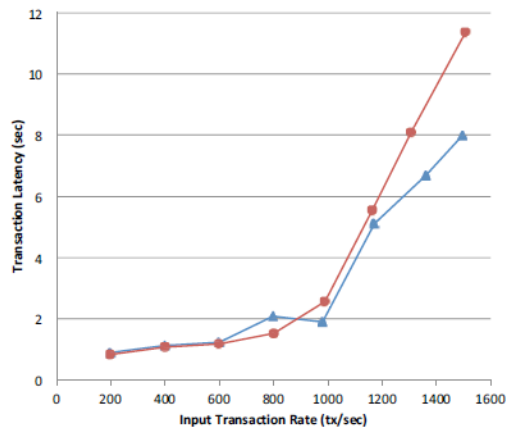
(a) Transaction latency



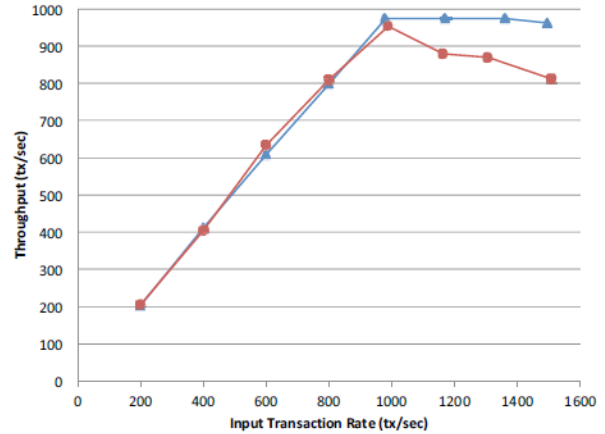
(b) Transaction throughput

Fig. 3: Latency and throughput measurements for varying input load

참고자료[1]의 하이퍼레저 패브릭의 성능 특성에 관해 연구한 논문입니다. Write, Mixed, Read, Null 상태에서 트랜잭션 지연율과 처리량을 도출하여 그래프로 나타냈습니다.



(a) Transaction latency



(b) Transaction throughput

Fig. 4: Effect of endorsement policy on transaction latency and throughput

endorsement peer가 싱글피어일 때와 더블 피어일 때 입력되는 트랜잭션을 늘려가면서 지연율과 처리량의 차이를 비교한 그래프입니다. 1000까지는 차이가 거의 없지만 1000tx/sec를 넘어가면 지연율이 늘어나고 처리량도 떨어지는 것을 확인할 수 있습니다.

Input Load (tx/s)	Batch Size	Throughput (tx/s)
1200	250	731
1200	500	801
1200	1000	1161
1200	1200	917
1400	500	869
1400	1400	1227
1600	500	812
1600	1600	1078

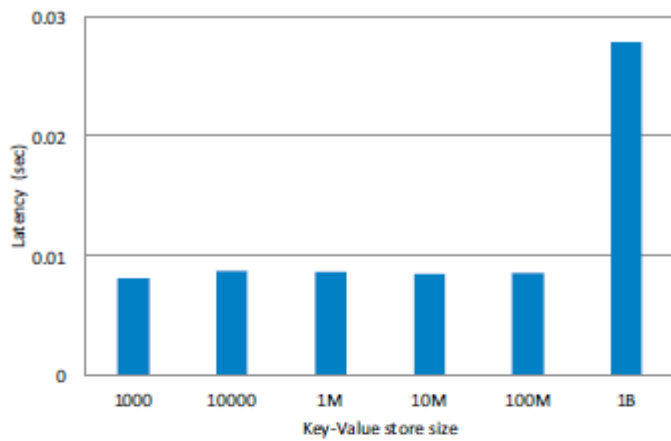
TABLE I: Effect of Orderer BatchSize on throughput

Orderer에 설정하는 배치크기(Batch)사이즈에 따라서 차이가 나는 결과를 나타낸 표입니다. 크기가 커짐에 따라서 처리량이 증가하다가 배치크기가 1000을 넘어가면 처리량이 감소하는 것을 볼 수 있습니다.

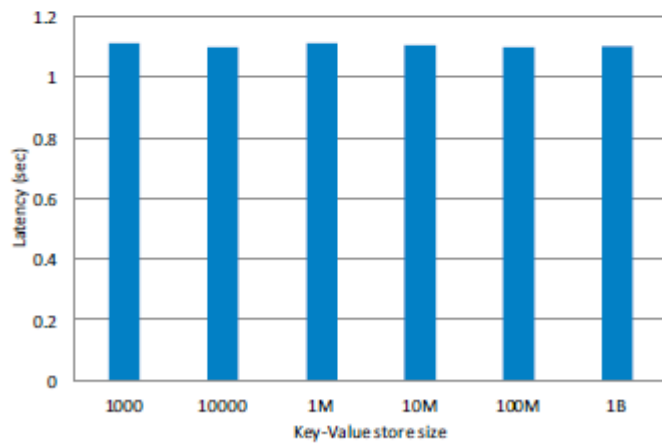
RW Set size (#entries)	Latencies (s)			
	c2e	e2o	o2v	Total latency
2 x 1000	0.238	0.008	0.120	0.366
2 x 0.3 Million	73.520	0.953	5.680	80.153
2 x 0.6 Million	147.650	1.887	12.630	162.167
2 x 1 Million	252.769	3.076	25.472	281.317

TABLE II: Transaction latency variation with RW set size

TABLE II에서는 크기가 다른 읽고-쓰기 집합에 따라서 지연시간 결과입니다. c2e는 보내는 데 걸린 시간입니다. e2o는 클라이언트가 연락하는 데 걸린 시간입니다. o2v는 트랜잭션을 정렬하는데 걸리는 시간입니다. 결과는 테이블2에서 나온것에 따라 c2e 전송하는데 보내는 시간이 가장 큰 것을 볼 수 있습니다.



(a) Read latencies



(b) Write latencies

읽기와 쓰기에서 걸리는 시간은 키-벨류 크기가 1십억이 될 때까지는 지연율이 크지 않은 것을 볼 수 있습니다. 쓰기의 경우에는 지연시간이 대체로 비슷한 것을 확인할 수 있습니다.

Payload Size(MB)	c2e(s)	e2o(s)	o2v(s)	Total latency(s)
1	0.114	0.101	0.221	0.436
10	0.902	0.981	1.627	3.510
20	1.791	1.958	3.045	6.794
30	2.747	2.899	4.936	10.582
40	6.432	3.861	10.778	21.071

TABLE III: Transaction latency with variable sized chaincode payloads

Payload Size(MB)	c2e (s)	e2o(s)	o2v(s)	Total latency(s)
1	0.132	0.149	0.296	0.577
10	1.092	1.454	2.442	4.988
20	2.117	2.896	4.585	9.598
30	5.323	4.335	10.497	20.155

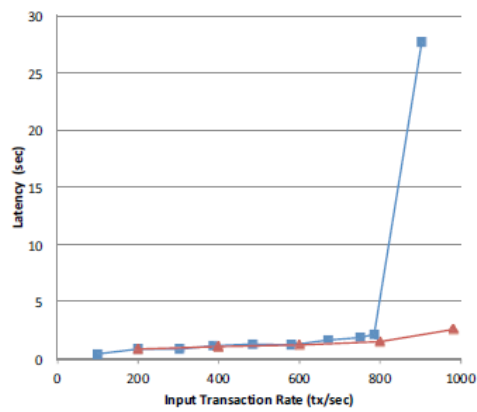
TABLE IV: Transaction latency with variable sized event payloads

테이블 3, 4는 체인코드 페이로드 크기에 따라 TABLE II에서 정의한 단계에서 지연시간을 구한 결과입니다. 트랜잭션을 정렬하는 시간에 많은 시간이 걸리는 것을 확인할 수 있습니다.

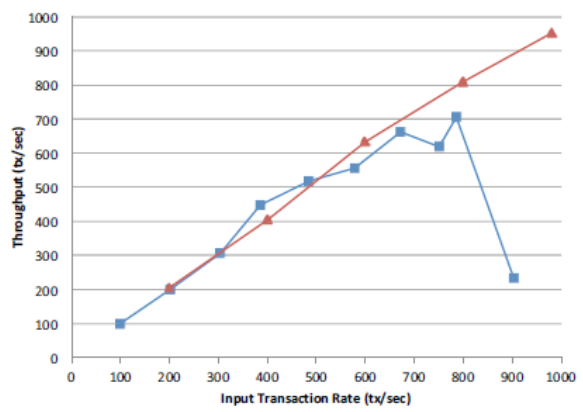
Call depth	Chaincodes		Overhead (%)
	Single (sec)	Multiple (sec)	
2	0.094	0.101	7.45
3	0.093	0.105	12.90
4	0.096	0.111	15.60

TABLE V: Inter-chaincode versus single chaincode transaction latencies

TABLE V에서는 싱글 체인코드의 호출과 다중 체인코드의 호출에 따른 오버헤드를 측정했습니다. 싱글체인 코드호출 보다 다중 체인코드 호출일 때 호출단계가 깊어질수록 늦어지는 것을 확인할 수 있습니다.



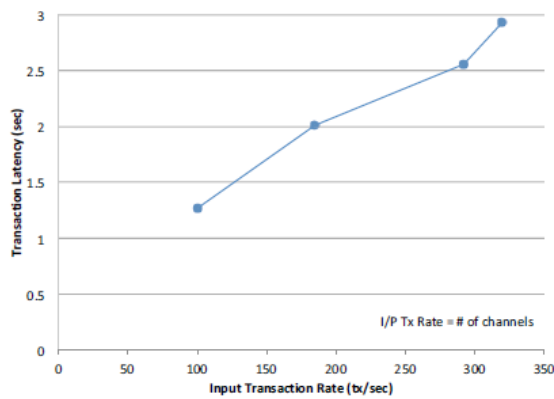
(a) Latency



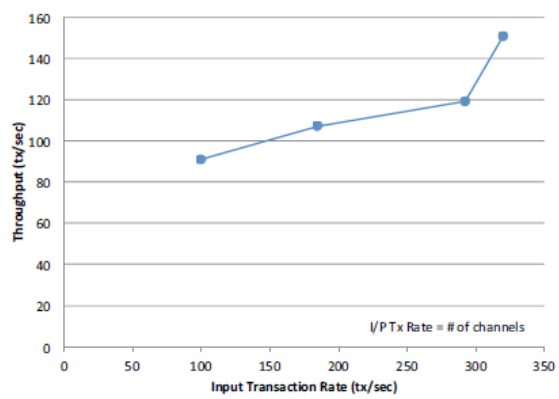
(b) Throughput

Fig. 6: Scaling the number of chaincodes

Fig. 6는 시스템의 지연시간 및 처리량을 보여줍니다. 다중체인코드 시 입력 트랜잭션이 800을 넘어가면 급격하게 지연시간이 늘어나는 것을 보입니다. 처리량도 800으로 다가갈수록 처리량이 떨어지다 반등하지만 900으로 갈수록 처리량은 감소하는 추세를 보이게 됩니다.

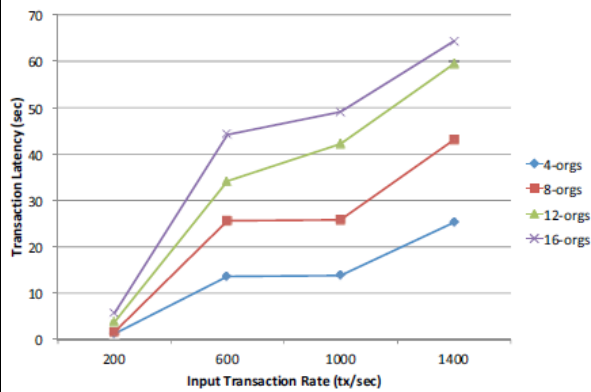


(a) Latency

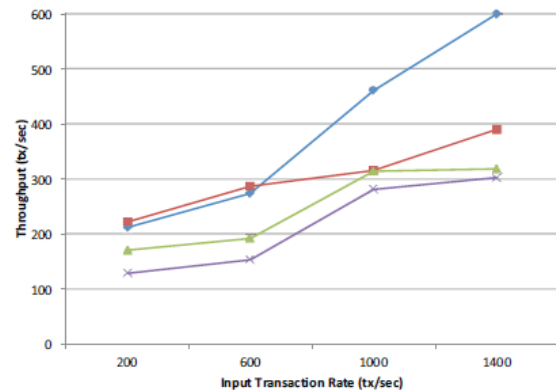


(b) Throughput

Fig. 7: Scaling the number of channels



(a) Latency



(b) Throughput

Fig. 8: Scaling the number of peers

Fig. 7은 채널의 개수에 따른 처리량과 지연시간을 보여줍니다. Fig 8.은 조직의 개수와 피어의 숫자에 따라 처리량과 지연율을 보여줍니다.

3. 차주 계획

- e2-enter-basic 작성 후 caliper-benchmark적용

4. 참고 자료

[1] Arati Baliga, Nitesh Solanki, Shubham Verekar, Amol Pednekar, Pandurang Kamat and Siddhartha Chatterjee "Performance Characterization of Hyperledger Fabric" 2018 Crypto Valley Conference on Blockchain Technology pp. 65-74