JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 25 No. 10, pp. 77-86, October 2020
https://doi.org/10.9708/jksci.2020.25.10.077

BR2K: A Replication and Recovery Technique Using Kubernetes for Blockchain Services

Min-Ho Kwon*, Myung-Joon Lee*

*Student, Dept. of Electrical/Electronic and Computer Engineering, University of Ulsan, Ulsan, Korea
*Professor, Dept. of Electrical/Electronic and Computer Engineering, University of Ulsan, Ulsan, Korea

[Abstract]

In this paper, we propose a technique for blockchain service replication and recovery using kubernetes(BR2K) that robustly executes blockchain services based on replication and supports systematic recovery in case of the service failure. Blockchain services are being developed and applied in various fields such as administration, finance, and medical systems based on the features of blockchain, such as decentralization, high security, and data integrity. In such areas where service continuity is important, it is necessary to provide robustness for execution of blockchain services, and a recovery plan for service failure is also required. To this end, BR2K provides an execution replication technique that systematically supports the sustainable execution of blockchain application services. Also, it introduces a robust container registry based on the blockchain service registry, systematically supporting the recovery of service failures by using it. In addition, Truffle, a blockchain service development framework, is extended to utilize the Kubernetes container management tool, and BR2K provides a technique for rapidly deploying blockchain services using the extended framwork.

▸ **Key words**: Blockchain Service, Service Robustness, Service Replication, Service Recovery, Service Registry

[요 약]

본 논문에서는 블록체인 응용서비스를 견고하게 실행하고, 이의 실패 시 체계적인 복구를 지원하는 BR2K(Blockchain application, Replication & Recovery technique using Kubernetes)기법을 제안한다. 블록체인 서비스는 블록체인의 특징인 탈중앙화, 높은 보안성, 그리고 데이터 무결성 등을 기반으로 행정, 금융, 그리고 의료 시스템 같은 다양한 분야에서 개발 및 적용되고 있다. 따라서 이와 같이 서비스의 연속성이 중요한 분야에서 블록체인 서비스 실행에 대한 견고성이 제공하는 것이 필요하며, 서비스 실패에 대한 복구 방안 또한 필요한 실정이다. 이를 위하여, BR2K는 블록체인 응용서비스의 지속 가능한 실행을 체계적으로 지원하는 실행 복제 기법을 제공한다. 또한, 블록체인 서비스 레지스트리 기반의 견고한 컨테이너 레지스트리를 소개하고 이를 이용하여 서비스 실패에 대한 복구를 체계적으로 지원한다. 더불어, 블록체인 서비스 개발 프레임워크인 트러플을 쿠버네티스 컨테이너 관리 도구를 활용할 수 있도록 확장하고, 이를 바탕으로 서비스를 신속하게 배포하는 기법을 제공한다.

▸ **주제어**: 블록체인 서비스, 서비스 견고성, 서비스 복제, 서비스 복원, 서비스 레지스트리

• First Author: Min-Ho Kwon, Corresponding Author: Myung-Joon Lee
*Min-Ho Kwon (alsgh458@gmail,com), Dept. of Electrical/Electronic and Computer Engineering, University of Ulsan
*Myung-Joon Lee (mjlee@ulsan.ac.kr), Dept. of Electrical/Electronic and Computer Engineering, University of Ulsan
• Received: 2020. 09. 16, Revised: 2020. 10. 05, Accepted: 2020. 10. 12.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

블록체인 응용서비스는 블록체인 기술의 보안성과 신뢰성을 기반으로 물류, 에너지, 금융, 공공 서비스 등 다양한 분야에서 빠르게 개발 및 적용되고 있다.[1-4, 14] 또한 블록체인 응용서비스가 증가함에 따라 블록체인 서비스 실행에 대한 견고성이나 실패 시의 복원에 대한 연구의 필요성 또한 중요하여지고 있다. 블록체인 서비스의 견고한 실행에 대한 연구에는 블록체인 응용서비스의 견고한 실행 기법[5], 쿠버네티스와 블록체인 서비스 레지스트리를 이용한 블록체인 서비스의 신속한 복제 실행 및 복구 기법[6] 등이 존재한다. 논문 [5]는 ETCD를[11] 기반으로 응용서비스에 대한 사용자의 요청을 체계적으로 복제하고 이를 일관성 있게 복제 노드들이 실행하기 위한 기법을 소개하고 있다. 그러나, 해당 기법을 다양한 분야의 블록체인 서비스에 실제로 적용할 때, 오직 한 번만 실행하여야 하는 블록체인 트랜잭션이나 요청의 처리가 성공적으로 중복되었으나 예외적인 상황으로 실행에 실패한 경우에 여러 문제가 발생할 수 있다.

본 논문에서는 복제된 요청을 실행할 때 발생할 수 있는 이러한 여러 가지 문제점을 체계적으로 해결하는 복제 실행과 복구 기법인 BR2K(Blockchain application, Replication & Recovery technique using Kubernetes)를 소개한다. 또한 블록체인 응용서비스가 재난상황에 의하여 전면적으로 실패하였을 경우, 체계적인 복구를 지원하는 복구 기법을 제안한다. 이를 위하여 [5]에서 제안된 블록체인 서비스 레지스트리를 확장하고 하버(Harbor)를[9] 이용한 견고한 컨테이너 레지스트리를 구성하여 활용한다. 이와 더불어, 복제 실행된 블록체인 응용서비스의 신속한 배포를 지원하기 위하여 이더리움 블록체인 개발 프레임워크인 트러플을[8] 확장하고 이를 통하여 개발된 블록체인 응용서비스를 분산 컨테이너 관리 도구인 쿠버네티스에[7, 15] 배포하는 기법을 소개한다.

본 논문의 구성은 1장 및 2장에서는 서론과 배경지식이 소개된다. 그리고 3장에서는 블록체인 응용서비스의 견고한 실행 및 체계적인 복구를 위한 BR2K 기법을 소개하고 4장에서는 제안한 서비스 복제 실행 기법의 유효성을 검증한다. 마지막 5장에서는 본 논문의 결론 및 추후 연구에 대하여 기술한다.

II. Background Knowledge

1. Robust execution scheme

블록체인 서비스의 견고한 실행 기법(Robust Execution scheme)은 블록체인 서비스의 실패에도 지속적으로 서비스를 제공하고 복구를 지원하기 위한 기법이다. 이 기법은 크게 서비스 복제 실행과 블록체인 서비스 레지스트리로 나뉜다. 서비스 복제 실행은 분산 환경을 구성하는 각 노드에 블록체인 서비스를 실행하고 상태를 복제하는 기법이다. 그림 1은 분산 환경에서 복제 실행된 블록체인 서비스의 요청 처리 순서를 나타낸다.

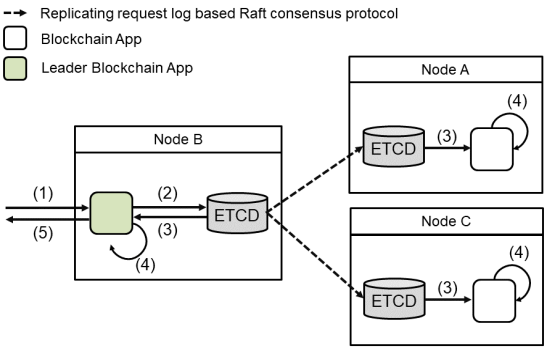

Fig. 1. Replication of Blockchain Application

서비스 복제 실행 기법이 적용된 블록체인 응용서비스는 Raft[10] 컨센서스 프로토콜 기반의 스토리지인 ETCD를 이용하여 서비스 상태를 복제한다. 그림 1처럼 각 노드에 실행된 블록체인 응용서비스는 ETCD를 이용하여 서비스 제공자 역할을 하는 하나의 리더 응용서비스를 선출한다. 만약 리더가 장애에 의해서 서비스를 제공하지 못하면, 이 기법에서는 새로운 리더를 선출하여 서비스를 지속적으로 제공되도록 한다.

(fig. 1, step 1) 그림 1과 같이 리더 응용서비스가 사용자에게 서비스에 대한 요청을 받을 때, 서비스 복제 작업이 시작된다. (step 2) 사용자 요청을 받은 리더 서비스는 자신의 로컬 노드에 있는 ETCD에 사용자 요청을 저장한다. 이때, 리더의 ETCD는 Raft 컨센서스 프로토콜 기반으로 각 노드에 사용자 요청을 복제한다. (step 3-4) 사용자 요청의 복제가 완료되면, 리더를 포함한 블록체인 응용서비스들은 복제된 요청을 처리하고 서비스 상태를 업데이트한다. (step 5) 이러한 과정이 완료되면, 리더는 요청에 대한 처리 결과를 사용자에게 응답한다.

블록체인 서비스의 견고한 실행 기법에는 블록체인 응용서비스의 복구를 지원하기 위한 서비스 레지스트리가 있다. 해당 서비스 레지스트리는 이더리움 스마트 컨트랙트[13] 기반으로 작성되어, 이더리움 블록체인에서 서비스 복구 정보를 안전하게 관리할 수 있고 사용자에게 서비스 위치 정보를 견고한 형태로 제공할 수 있다.

2. Container and Kubernetes

컨테이너(Container)는[12] 기존의 가상화 기술의 단점을 극복한 경량 가상화 기술이다. 컨테이너 기술은 임의의 환경에서도 제약 없이 신속하게 실행하기 위하여, 응용서비스 실행에 필요한 모든 정보를 컨테이너 이미지라는 파일 형태로 빌드하고 이 이미지를 기반으로 응용서비스를 신속하게 실행한다.

쿠버네티스(Kubernetes)는 분산 환경에서 컨테이너를 관리하기 위한 도구이다. 쿠버네티스를 이용하면 분산 환경에서 컨테이너를 빠르게 배포 및 확장할 수 있으며, 배포된 컨테이너 관리를 자동화할 수 있다. 쿠버네티스의 구성은 마스터 노드와 워커 노드로 구성된다. 마스터 노드(Master Node)는 쿠버네티스 클러스터를 관리하는 컴포넌트들이 실행되는 장소이다. 워커 노드(Worker Node)는 컨테이너의 실행 환경을 제공하고 각 노드의 컨테이너를 관리하는 컴포넌트들이 실행되는 장소이다.

쿠버네티스에서 컨테이너의 배포는 최소 배포 단위인 파드(Pod)로 이루어진다. 파드 형태를 이용하면 같은 로컬 네트워크나 디스크를 공유하는 다수의 컨테이너도 한번에 배포할 수 있다. 쿠버네티스에서 컨테이너로 구성된 파드가 배포되는 과정은 다음과 같다.

1) 쿠버네티스에서 모든 통신을 담당하는 마스터 노드의 컴포넌트인 API 서버에게 파드 배포를 요청한다.

2) API 서버는 마스터 노드의 컴포넌트인 스케줄러에게 파드가 실행될 워커 노드 선정을 요청한다.

3) 해당 스케줄러는 스케줄링 원칙에 따라 컨테이너가 실행될 적절한 워커 노드들을 선정하여 API 서버에게 전달한다.

4) API 서버 컴포넌트가 선정된 워커 노드들에게 컨테이너의 실행을 요청하여, 쿠버네티스 환경에서 파드가 배포된다.

3. Truffle framework

트러플(Truffle)은 이더리움 기반의 블록체인 서비스의 손쉬운 개발을 지원하는 프레임워크이다. 일반적인 이더리움 블록체인 개발 핵심은 블록체인상에서 특정 조건에 따라 기능을 수행하는 스마트 컨트랙트(Smart contract)를[13] 개발하고 블록체인에 배포 및 테스트하는 것이다. 트러플 프레임워크를 이용하면 이더리움의 스마트 컨트랙트에 대한 컴파일, 배포, 테스트에 대한 복잡성을 추상화하여 빠르게 작업을 수행할 수 있다. 또한 트러플을 이용하면 배포된 스마트 컨트랙트의 트랜잭션, 상태 등을 모니터링할 수 있으며, 다른 개발자가 유용한 모듈, 라이브러리 등을 묶어 놓은 트러플 박스를 이용하여 보다 빠르게 개발할 수 있다.

III. BR2K scheme

본 장에서는 블록체인 응용서비스의 견고한 실행, 신속한 배포 및 체계적인 복구를 지원하는 BR2K 기법을 제안하고, 해당 기법을 손쉽게 사용하기 위한 트러플 프레임워크의 확장에 대해서 설명한다.

1. Replication method

본 논문에서는 BR2K 기법을 이용하여 다중으로 복제된 블록체인 응용서비스를 BR2K 서비스로 명명한다. 이 서비스를 구성하는 블록체인 응용서비스의 역할에는 리더(Leader), 팔로워(Follower), 레지스터(Register)가 있으며, 하나의 BR2K 서비스는 리더 응용서비스와 다수의 팔로워 응용서비스로 구성된다. 또한 하나의 응용서비스가 실행되는 노드에는 ETCD도 함께 실행된다. 그림 2는 BR2K 서비스을 구성하는 블록체인 응용서비스의 역할 전환을 나타낸다.

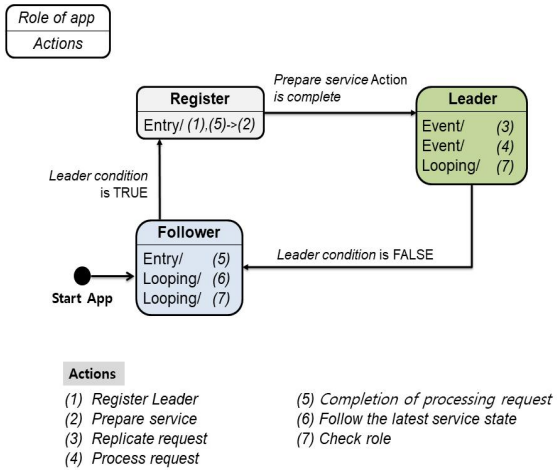

Fig. 2. Role transition of blockchain application

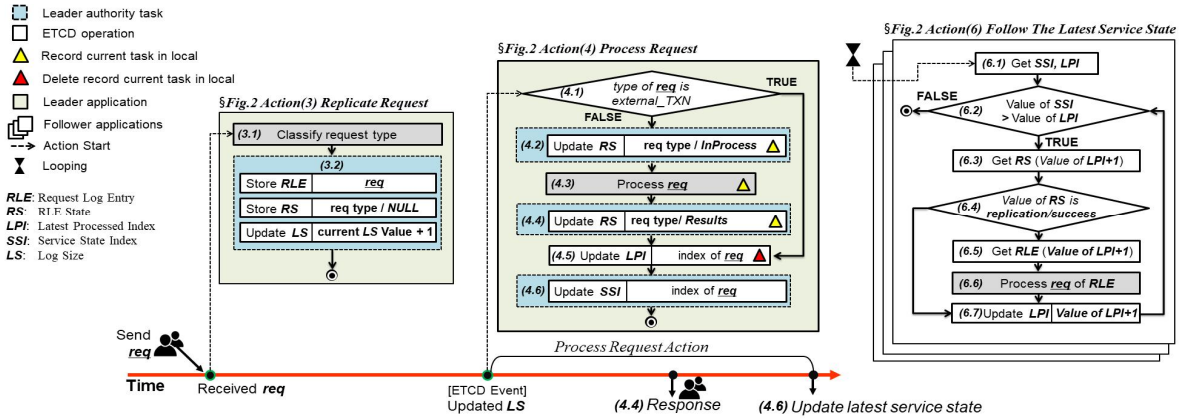

Fig. 3. Leader & follower actions for state replication of blockchain application

리더 역할을 수행하는 응용서비스는 사용자에게 서비스를 제공하고 블록체인 응용서비스의 상태를 업데이트한다. 팔로워 응용서비스는 리더가 업데이트한 최신 서비스 상태를 지속적으로 동기화하는 역할이다. 그림 2의 레지스터 응용서비스는 리더 역할을 수행하기 전에, 블록체인 서비스 레지스트리에 새로운 서비스 접근 위치를 업데이트하고 이전 역할에서 업데이트하지 못한 최신 서비스 상태를 동기화하는 역할이다.

블록체인 응용서비스의 역할은 작업 과정 절차인 액션(Action)이라는 단위로 작업을 수행한다. BR2K 서비스가 시작하면, 모든 블록체인 응용서비스의 역할은 팔로워가 된다. 팔로워 응용서비스는 리더 역할을 수행할 수 있는 조건인 *Leader condition* 을 주기적으로 검사하는 *Check role* 액션이 수행된다. 이 액션에서 *Leader condition* 은 응용서비스에 연결된 ETCD의 상태가 리더라면 TRUE가 되고, 아니라면 FALSE가 된다. *Leader condtion* 이 TRUE인 팔로워는 레지스터 역할로 전환되어 서비스 준비 과정을 거친 뒤, 사용자에게 서비스를 제공하는 리더 응용서비스가 된다.

그림 3은 블록체인 응용서비스의 상태 복제를 위한 리더 및 팔로워 액션을 나타낸다. 리더의 *Replicate Request* 액션은(§fig. 2,3) 사용자 요청을 리더가 바로 처리하지 않고, 해당 요청을 ETCD에 저장하여 모든 응용서비스에 복사하는 작업을 수행한다. 하나의 사용자 요청에 대한 *Replicate Request* 액션의 작업 절차는 다음과 같다.

[(3.1) in fig. 3] 모든 요청에 대한 복제 실행을 피하기 위하여, 사용자 요청의 종류를(§fig. 4) RESTful API 기반으로 분류한다.

Secetion 1. Data stroed in ETCD
(Data format is key-value)

RLE(Request Log Entry)	Client request to store in the form of index Key: (Index), Value: request
RS(RLE State)	(request type, processing state) of *RLE* Key: "RS-"(index), Value: (Request Type, *State type) *State type (*Null, InProcess, Sucess, Fail, Ignore*)
SSI(Service State Index)	Index representing the latest service state Key: "Service_State_Index", Value: (Index)
LPI(Latest Processed Index)	Index of *RLE* last processed by app Key: ID of the app on each node Value: (Index)
LS(Log Size)	Number of *RLE* stored in ETCD Key: "Log_Size", Value: (Number of request log entries)
CL(Current Leader)	ID of current leader application Key: "Current_Leader", Value: (Number of request log entries)

Secetion 2. Request type

Replication	Request that be processed by all redundant Blockchain services
Normal	Request that must only be executed onced
External_TXN	Request to record blockchain transactions occurring at the client side

Fig. 4. Data & Request type

[(3.2) in fig. 3] 리더는 분류된 요청이 BR2K 서비스에서 몇 번째 요청인지 나타내는 인덱스 값을 할당한다. 인덱스 값은 현재 사용자 요청의 총 개수를 나타내는 *LS*(Log Size, §fig. 4)의 값에 1 증가 시켜 할당한다. 요청에 대한 인덱스 할당이 완료되면, 해당 요청을 사용자 요청 로그인 *RLE* (Request Log Entry, §fig. 4) 형태로 ETCD에 저장

한다. 또한 해당 요청의 처리 과정에서 발생할 수 있는 장애나 실패를 극복하기 위하여, 해당 요청의 처리 상태 값과 종류를 *RS* (RLE State, §fig. 4) 형태로 ETCD에 저장한다. *RS* 의 초기 상태 값은 *Null* (§fig. 4)로 저장된다. 더불어 저장된 요청의 개수가 ETCD 내에서 증가하기 때문에, *LS* (Log Size, §fig. 4)의 값도 1 증가 시켜 업데이트한다.

Replicate Request 액션으로 인하여 ETCD에서 *LS* 의 업데이트가 완료되면, ETCD에서 *LS* 에 대한 업데이트 이벤트가 발생한다. 이는 ETCD에 의해, 현재 *LS* 의 값을 인덱스로 하는 사용자 요청이 복제되었음을 의미한다. 이 이벤트를 감지한(§Table. 1, line 16) 리더 응용서비스는 해당 요청을 처리하고 응답하는 작업인 *Process Request* 액션을(§fig. 2) 수행한다. 이 액션의 과정은 다음과 같다.

[(4.1) in fig. 3] 리더가 처리해야 하는 사용자 요청의 종류가 *External_TXN* (§fig. 4) 타입인지 확인한다. 이 요청의 종류가 *External_TXN* 이 아니라면, 리더 응용서비스는 이 요청에 대한 처리 작업을 수행한다.

[(4.2-4) in fig.3] 리더는 해당 요청의 처리 중에 발생할 수 있는 응용서비스의 중지나 역할 전환에 대응하기 위하여, 해당 요청에 대한 *RS* 의 처리 상태를 *InProcess* (§fig. 4) 상태로 업데이트한다. *RS* 의 상태에 대한 업데이트가 완료되면, 리더는 해당 요청에 대한 처리 작업을 수행하고 처리 결과에 대한 응답을 사용자에게 보낸다. 리더는 요청에 대한 처리 결과를 *Sucess*, *Fail* (§fig. 4)로 구분하며, 이 결과를 해당 요청에 대한 *RS* 의 처리 상태 값으로 업데이트한다. 이 작업으로 인하여 그림 3의 (6.4)처럼 팔로워들은 처리 결과가 실패가 아니고 타입이 *Replication*인(§fig. 4) 요청에 대해서만 처리 작업을 수행하여, 불필요한 요청의 처리를 피하게 된다.

[(4.5-6) in fig. 3] 리더는 마지막으로 응용서비스가 처리한 요청의 인덱스를 나타내는 *LPI* (§fig. 4, Latest Processed Index)을 해당 요청의 인덱스로 업데이트한다. 또한, 리더는 블록체인 서비스의 최신 상태를 의미하는 *SSI* (§fig. 4, Service State Index)도 해당 요청의 인덱스로 업데이트한다. 이 *SSI* 의 업데이트로 인해 이벤트가 발생하고, 이를 감지한(§Table. 1, line 27) 팔로워들은 최신 서비스의 상태를 따라가는 *Follow The Latest Service State* 액션이(§fig. 2,3) 실행된다.

그림 3에 *LS*, *SSI*, *RS* 의 업데이트와 *RLE* 의 저장 작업은 리더 응용서비스만 작업할 수 있는 리더 권한 작업 (Leader authority task)이다. 응용서비스가 리더 권한을 가지기 위해서는 *Leader condition* 이 TRUE 이고 현재 리더를 나타내는 *CL*(Current Leader, §fig. 4)의 값을 해당 응용서비스의 식별 값으로 업데이트해야 한다. 리더 권한 작업은 ETCD에서 제공하는 기능인 조건 트랜잭션 기반으로 동작하여, ETCD가 이 작업을 수행하려고 하는 응용서비스의 리더 권한을 검사한다. 만약 이전 리더가 새로운 리더가 등장했음에도 리더 권한 작업을 수행하려고 한다면, ETCD에 의하여 해당 작업은 거절되고 이전 리더의 역할은 팔로워로 전환된다.

Table 1. Pseudocode for [§fig. 2 Action 2, 5]

Prepare service & Check for pending process

```
ETCD OPERATION
GET(KEY)                                  /*get VALUE of KEY*/
STORE(KEY, VALUE)                         /*store VALUE of KEY*/
UPDATE(KEY, VALUE)                        /*update VALUE of KEY*/
WATCH(KEY)      /*watch for updates to the value of KEY*/

Action(2) Prepare Service (§Fig. 2)
 1: lpi ← GET(appID)      /*lpi is Latest Processsed Index*/
 2: logSize ← GET('Log_Size')
 3: WHILE lpi < logSize
 4:   curIndex ← lpi+1
 5:   rs ← GET('RS/'+curIndex)          /*rs is RLE States*/
 6:   IF state of rs is Null
 7:       UPDATE('RS/'+curIndex, Ignore)
 8:   ELSE
 9:     IF type of rs is replication & state of rs is not Fail
10:       request ← GET(curIndex)
11:       result ← PROCESS request
12:       IF processing state of rs is InProcess
13:         UPDATE('RS/'+curIndex, result)
14:   UPDATE('Latest_Processed_Index', curIndex)
15:   UPDATE('Service_State_Index', curIndex)
16: WATCH('Log_Size')

Action(5) Check for pending process(§Fig. 2)
17: IF there is a record file of current work in local
18:   (curIndex, curWork) ← READ record
19:   IF curWork is (4.2 task) of process request Action(4)
20:     request ← GET(curIndext)
21:     PROCESS request
22:   IF curWork is (4.3 task) of process request Action(4)
23:     request ← GET(curIndext)
24:     ROLLBACK request
25:     PROCESS request
26:   UPDATE('Latest_Processed_Index', curIndex)
27: WATCH('Service_State_Index')
```

Prepare service 액션은(§fig. 2,3) 레지스터 응용서비스가 새로운 리더가 되어 서비스를 제공하기 전에, 최신 서비스 상태를 동기화하기 위한 작업이다. 해당 액션에서는 불필요한 요청 처리를 피하기 위하여, 이전 리더에 의해서 복제된 요청을 처리 상태와 타입에 따라 구분지어 처리한다.

[line 6-7 in table.1] 리더가 될 레지스터 응용서비스는 *Null* 상태의 요청에 대해서는 처리 상태를 *Ignore* (§ fig. 4)로 업데이트하고, 해당 요청에 대해서 처리하지 않는다.

[line 9-13 in table.1] 레지스터 응용서비스는 오직 요청 타입이 *Replication* (§fig. 4)이고 처리 상태가 *Fail* (§fig. 4)이 아닌 요청에 대해서만 처리한다. 특히, 처리 상태가 *InProcess* (§fig. 4) 인 요청은 이전 리더가 처리하고 있는 중에 중단된 요청으로, 현재 레지스터 응용서비스는 이전 리더가 해당 요청에 대해서 어떤 부분까지 처리하였는지 파악할 수가 없다. 이러한 이유로 레지스터가 *InProcess* 인 요청을 처리하지 않고 넘어간다면, 현재 레지스터와 이전 리더의 상태가 서로 달라질 수 있다. 그러므로 레지스터는 *InProcess* (§fig. 4)인 요청을 넘어가지 않고 처리하며, 처리 결과를 해당 요청의 처리 상태에 (processsing of RS) 업데이트한다.

Completion of processing request 액션은(§fig. 2,3) 이전 리더 응용서비스가 요청을 처리하는 중에 역할 전환이 발생하여, 해당 응용서비스에 중지된 작업이 있는지 검사하기 위한 액션이다. 이 액션에서는 리더의 *Replicate Request* 액션 과정에서 (§ fig. 2,3, Record current task in local) 생성된 현재 작업 정보 파일이 있다면, 중지된 작업이 있다고 판단한다. 중지된 작업에 대한 요청은 새로운 리더가 *Prepare service* 액션에서 건너뛰지 않고 처리하기 때문에, 중지된 작업이 있는 이전 리더 응용서비스는 해당 요청 처리에 대한 완결성을 가져야한다. 중지된 작업의 종류는 *RS* 의 처리 상태를 *InProcess* 로 업데이트하는 작업, 요청을 처리하는 작업 그리고 요청 처리 결과를 *RS* 의 처리 상태로 업데이트하는 작업으로 구분된다.

[line 19-21,26 in table.1] 중지된 작업의 종류가 *RS*의 처리 상태를 *InProcess*로 업데이트하는 작업이면, 해당 응용서비스는 중지된 작업의 요청에 대해서 처리하고 *LPI* 의 값을 처리한 요청의 인덱스로 업데이트한다.

[line 23-26 in table.1] 중지된 작업의 종류가 요청을 처리하는 작업이면, 해당 요청에 대하여 설정된 롤백(Rollback)을 수행하여 응용서비스의 상태를 요청을 처리하기 전의 상태로 되돌린다. 그리고 응용서비스는 해당 요청에 대해서 다시 처리 작업을 수행하고 *LPI* 의 값을 처리한 요청의 인덱스로 업데이트한다.

[line 26 in table.1] 중지된 작업의 종류가 요청 처리 결과를 *RS* 의 처리 상태로 업데이트하는 작업이면, 중지된 작업의 요청에 대한 인덱스 값만 *LPI* 의 값으로 업데이트한다.

2. method for rapid deployment

본 절에서는 BR2K 서비스를 분산 환경에서 일관성있는 형태로 신속하게 실행하기 위하여, 쿠버네티스 기반의 배포 기법과 이를 위한 트러플 프레임워크 확장 기법에 대하여 설명한다.

BR2K 서비스를 쿠버네티스 환경에서 구성하기 위하여, 블록체인 응용서비스와 ETCD를 컨테이너 이미지로 빌드하고 쿠버네티스의 워커 노드에 라벨링 작업을 한다. 라벨링은 그림 5의 (1)과 같이 키-값으로 구성된 고유한 라벨을 쿠버네티스의 각 워커 노드에게 할당하는 작업이며, 이 작업을 완료한 쿠버네티스에서는 서비스를 배포할 때 서비스 실행 환경인 워커 노드를 선택하는 기능을 사용할 수 있다. 쿠버네티스 기반의 배포 기법은 이러한 기능을 사용하여, 하나의 노드에 블록체인 응용서비스와 이를 위한 ETCD를 함께 실행하는 BR2K 서비스를 쿠버네티스에 배포한다.

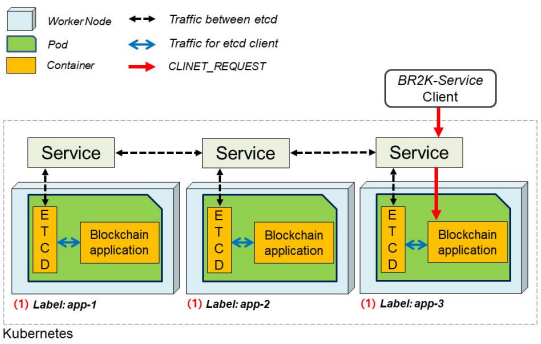

Fig. 5. BR2K service running on kubernetes

그림 5는 쿠버네티스에 배포된 BR2K 서비스를 나타낸다. 배포된 BR2K 서비스에서는 블록체인 응용서비스 컨테이너와 ETCD 컨테이너를 묶은 파드가(§2.2) 하나의 워커 노드에 실행된다. 하나의 파드에는 서비스(Service) 오브젝트가 연결된다. 서비스 오브젝트는 외부에서 접근 가능한 고정적인 엔드포인터를 가지고 있어, 사용자 요청을 파드에게 전달하는 역할을 수행하고 ETCD 클러스터를 구성할 때 사용된다. 그림 5와 같이 하나의 파드에 실행되는 ETCD는 로컬 네트워크 환경에서만 작업 요청을 받도록 설정하여, 오직 같은 파드에서 실행되는 블록체인 응용서비스만 이용할 수 있다.

쿠버네티스 기반의 배포 기법을 실제 개발 환경에서 손쉽게 이용하기 위하여, 이더리움 블록체인 서비스 개발 프레임워크인 트러플을(§2.3) 확장한다. 확장된 트러플은 이더리움 블록체인 서비스 개발, 컨테이너 이미지 빌드 그리고 쿠버네티스에 BR2K 서비스 배포 등의 기능을 제공하며, 모든 기능들은 커맨드 라인 인터페이스 기반으로 동작한다. 그림 6은 확장된 트러플의 구조를 나타낸다.

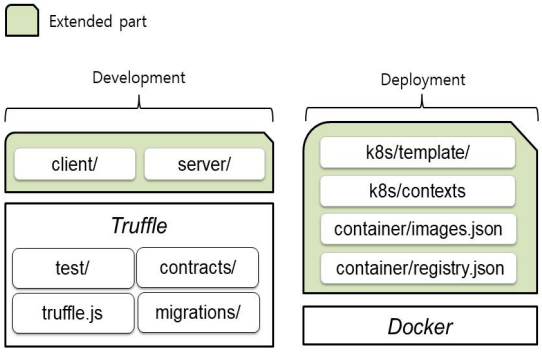

Fig. 6. structure of extended truffle framework

확장된 프레임워크는 그림 6와 같이 트러플과 컨테이너 플랫폼인 도커 기반으로 확장된다. 이 프레임워크 구조는 BR2K 서비스에 대한 개발과 배포로 나뉜다. 그림 6의 개발(Development) 파트는 트러플 프레임워크 기반으로 이더리움의 스마트 컨트랙트 배포 및 테스트, 이더리움 블록체인 응용서비스 개발의 복잡성을 (§fig.6, client/ & server/) 추상화시켜준다. 이 프레임워크의 배포(Deployment) 파트에서는 아래와 같은 작업을 수행할 수 있다.

- 블록체인 응용서비스의 컨테이너 이미지 빌드 및 빌드된 이미지 관리 (§fig.6, images.json)
- 컨테이너 레지스트리에 빌드된 컨테이너 이미지 저장 및 관리 (§fig.6, registry.json)
- 쿠버네티스에 BR2K 서비스 배포(§fig.6, contexts)

본 프레임워크의 배포 파트에서 쿠버네티스와 관련된 작업을 수행하려면, 쿠버네티스의 관리자에게 할당받은 컨텍스트(Context) 정보가 필요하다. 이 컨텍스트는 서비스 배포자가 쿠버네티스에서 사용 가능한 리소스 정보들이 명세되어 있으며, 오직 쿠버네티스 관리자에 의해서 만들어진다.

그림 7은 컨텍스트를 이용하여 쿠버네티스에 배포한 BR2K 서비스를 나타낸다. 컨텍스트는 쿠버네티스의 위치(Location), 쿠버네티스에서 논리적인 작업 영역인 네임스페이스(Namespace), 사용권한 정보인 액세스 토큰(Access Token)인 세 가지 정보의 조합으로 식별된다. 또한 컨텍스트는 사용 가능한 워커 노드 수, 외부에서 접근 가능한 고정 엔드포인트, 서비스 저장 공간인 불륨 등의 클러스터 자원(Cluster Resource) 정보도 포함한다.

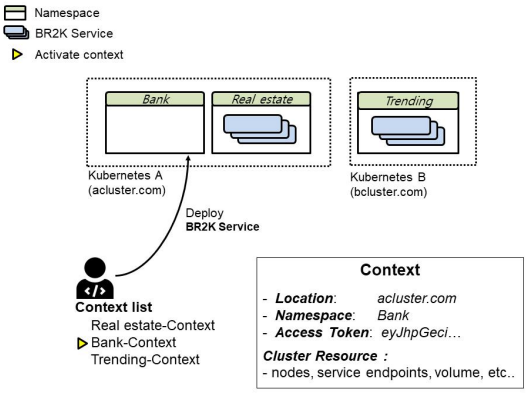

Fig. 7. Deploying BR2K Service on Kubernetes using context

본 프레임워크를 사용하는 서비스 배포자는 쿠버네티스의 관리자로부터 한 개 이상의 컨텍스트 정보를 할당받고 그림 7과 같이 프레임워크의 컨텍스트 리스트에 저장할 수 있다. 또한 서비스 배포자는 컨텍스트 리스트에 있는 컨텍스트들 중 하나의 컨텍스트를 활성화시키고 본 프레임워크에서 구현된 배포 커맨드를 이용하여 쿠버네티스에 BR2K 서비스를 신속히 배포할 수 있다.

3. Recovery Method

본 절에서는 컨테이너 이미지를 안전하게 관리하기 위한 오픈 소스 컨테이너 레지스트리 하버의 구성과 블록체인 서비스 레지스트리를 이용한 BR2K 서비스 실패에 대한 체계적인 복구 지원 기법에 대해 설명한다.

하버는 마이크로서비스 아키텍처로, 하나의 서비스를 제공하는 단일 컴포넌트의 집합으로 구성된다. 하버는 크게 접근 권한, 이미지, 복제, 프로젝트 관리 등의 메인 서비스를 담당하는 코어(Core), 로그를 관리하는 로그 컬렉터(Log Collector), 유저 인터페이스(UI), 이미지 저장소인 레지스트리(Registry)로 구성된다.

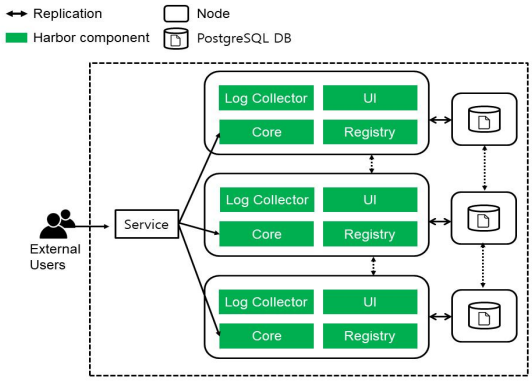

Fig. 8. Architecture of harbor deploying in kubernetes

본 기법에서는 BR2K 서비스의 컨테이너 이미지를 보관하는 하버의 가용성을 높이기 위하여, 쿠버네티스 기반으로 하버를 구축한다. 그림 8은 쿠버네티스에 배포된 컨테이너 레지스트리 하버의 아키텍처이다. 하버 컴포넌트들을 쿠버네티스의 각 노드에 실행하여 다중화하고, 하버의 인증 정보, 이미지 메타 정보 등을 관리하는 PostgreSQL 데이터베이스도 실행하여 하버의 가용성을 높인다. 또한 쿠버네티스에서 외부 요청에 대한 로드밸런서 및 외부 엔드포인트 역할을 수행할 수 있는 서비스 오브젝트를 각 노드에 있는 하버 컴포넌트인 코어랑 맵핑시켜, 외부 사용자가 하버를 이용할 수 있도록 한다.

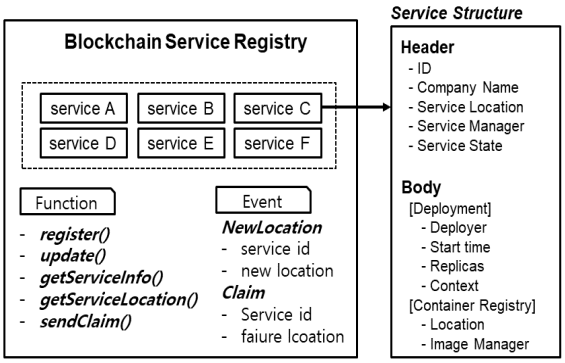

Fig. 9. Structure of blockchain service registry

그림 9은 이더리움 스마트 컨트랙트로 설계된 블록체인 서비스 레지스트리의 구조를 나타낸다. 블록체인 서비스 레지스트리에서 저장되는 서비스 구조(Service Structure)는 헤더(Header)와 바디(Body)로 나뉜다. 서비스의 헤더에는 서비스 접근 위치, 관리자 및 서비스 상태와 같은 서비스 메타 정보가 저장된다. 서비스의 바디에는 배포자, BR2K 서비스 배포에 사용된 컨텍스트(§3.2), 블록체인 응용서비스의 복제수 그리고 BR2K 서비스의 컨테이너 이미지를 관리하는 하버의 위치와 같은 서비스 복구 정보가 저장된다. 이와 더불어 [5]에서 개발된 서비스 복구 요청 기능(§fig. 9, *sendClaim*), 현재 서비스 위치 얻기(§fig. 9, *getServiceLocation*) 그리고 서비스 위치 업데이트와 복구 요청을 알리는 이벤트 기능(§fig. 9, *NewLocation & Claim*)과 같은 기존의 서비스 레지스트리의 모든 기능을 제공한다.

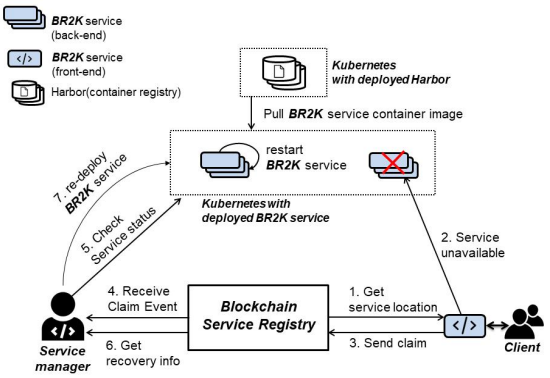

Fig. 10. Recovery process using blockchain service registry

그림 10은 블록체인 서비스 레지스트리를 이용한 복구 절차를 나타낸다. BR2K 서비스는 그림 10과 같이 블록체인 서비스 레지스트리에서 서비스 위치를 가져와 서비스를 제공한다. 만약, BR2K 서비스가 예정된 서비스를 제공하지 못할 경우, 사용자는 해당 서비스 레지스트리에 서비스 복구 요청 기능인 *sendClaim* 을 사용하여 관리자에게 복구 요청을 보낸다.

서비스 사용자가 서비스 복구 요청 기능을 사용하면 블록체인 서비스 레지스트리에서 *Claim* 이벤트가 발생하고, 이 이벤트를 감지한 서비스 관리자는 쿠버네티스에 배포된 BR2K 서비스의 상태를 확인한다. 관리자는 해당 서비스의 장애나 중지가 있다고 판단되면, 블록체인 서비스 레지스트리에 저장된 서비스 복구 정보를 가져오고 이를 이용해 BR2K 서비스를 쿠버네티스에 재배포한다. 이때, BR2K 서비스가 재배포되는 과정에서 서비스 복구 정보에 있는 컨테이너 레지스트리인 하버에 접근하여 BR2K 서비스의 컨테이너 이미지를 가져오고 해당 서비스를 재실행하게 된다.

IV. Test

BR2K 기법의 서비스 복제 실행의 유효성을 검증하기 위하여, 이 기법을 적용한 테스트 응용서비스를 개발하여 테스트한다. 테스트 응용서비스는 네트워크 장애 시에도 응용서비스의 상태 복제의 정확성을 확인하기 위하여 사용자 요청의 식별 아이디를 기록하는 로그 서버이다. 그림 11은 테스트 환경을 보여주며 분산 환경을 구성하는 각각의 노드에 하나의 ETCD와 테스트 응용서비스를 실행한다.

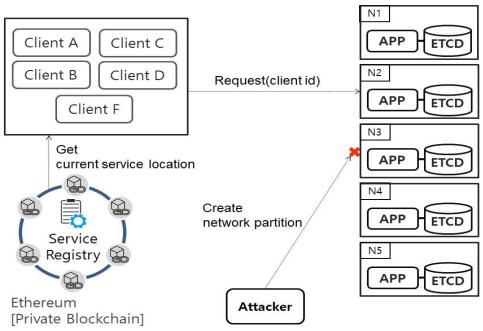

Fig. 11. Test environment

그림 11에서 Attacker는 특정 주기마다 응용서비스가 실행되는 노드에 20초간 네트워크 분할을 발생시키는 프로세스이다. 테스트 응용서비스에게 요청을 보내는 사용자는 5명이며, 각자 고유한 식별 아이디를 가지고 있다. 이 사용자들은 블록체인 서비스 레지스트리에서 서비스 제공 위치를 확인하여 사용자 요청을 보내게 된다. 만약 사용자가 현재 이용 중인 서비스에 요청을 보내는 것이 실패한다면 블록체인 서비스 레지스트리에서 다시 서비스 제공 위치를 가져와 요청을 보낸다. 본 논문의 테스트 과정은 다음과 같다.

(1) 각 사용자는 총 10,000번의 요청을 초당 2번의 속도로 테스트 응용서비스에게 보낸다. 이때 사용자 요청은 각 사용자의 고유한 식별 아이디를 포함하고 있다. 반면에 Attacker는 사용자가 요청을 보내는 동안 랜덤한 주기마다 네트워크 장애를 발생시킨다.

(2) 사용자 요청을 받은 테스트 응용서비스는 요청에 대해서 처리하고 응용서비스의 서비스 상태를 복제한다. 이때 테스트 응용서비스는 사용자 요청에 포함되어 있는 사용자 고유의 식별 값을 사용자 요청 로그 리스트에 저장한다. 이러한 작업이 완료되면, 사용자에게 해당 요청이 저장된 리스트의 인덱스 값을 응답한다.

(3) 사용자는 요청에 대한 응답을 받으면, 응답에 있는 인덱스 값을 사용자 응답 로그 리스트로 저장한다. 모든 사용자가 10,000번의 요청을 완료하면, 복제 실행된 테스트 응용서비스와 모든 사용자가 가지고 있는 리스트 사이에서 누락된 부분이 없는지 검사하여 서비스 복제가 정확하게 이루어졌는지 확인한다.

표 2의 테스트는 리더 테스트(Leader test)와 랜덤 테스트(Random test)가 있다. 리더 테스트는 특정 범위의 랜덤한 주기마다 리더 응용서비스가 실행되는 노드에만 네트워크 분할을 발생하는 테스트이다. 표 2의 테스트는 리더 테스트(Leader test)와 랜덤 테스트(Random test)가 있다.

Table 2. Test result

	Leader Test	Random Test
Frequency of network partition	120~140sec	60~120sec
# of leader changes	52	32
# of *Ignore* states (Out of 50,000 requests)	9,518	2,510
# of *Success* states (Out of 50,000 requests)	40,480	47,489
# of *InProcess* states (Out of 50,000 requests)	2	1
State replication	success	success

리더 테스트는 특정 범위의 랜덤한 주기마다 리더 응용서비스가 실행되는 노드에만 네트워크 분할을 발생하는 테스트이다. 이 테스트에서는 리더가 요청을 처리하는 중에 네트워크 분할이 발생해도 응용서비스의 서비스 상태 복제가 정확하게 이루어지는 지 확인한다. 랜덤 테스트는 임의의 노드를 선택해 네트워크 분할을 일으키는 테스트이다. 이 테스트에서는 네트워크 분할을 극복한 응용서비스가 최신 서비스 상태를 정확하게 동기화 할 수 있는 지 확인한다. 두 테스트의 결과는 각 응용서비스가 가지고 있는 사용자 요청 로그 리스트가 서로 일치하는 지, 모든 사용자 응답 로그 리스트의 합이 사용자 요청 로그 리스트와 일치하는 지 확인한다. 표 2에 있는 각 테스트 결과에서 볼 수 있듯이, 빈번한 리더의 변경과 요청 처리 중에 장애가 발생하여도(table 2, # of *InProcess* states) 상태 복제가 정확하게 이루어졌으며 지속적으로 서비스가 제공할 수 있음을 확인하였다.

V. Conclusions

본 논문에서는 블록체인 응용서비스의 견고한 실행을 위한 서비스 복제 실행 및 체계적인 복구를 지원하는 BR2K 기법을 소개하였다. BR2K 기법은 체계적인 서비스 복제 실행을 통하여, 블록체인 응용 서비스가 네트워크 실패 시에도 지속적으로 정확한 서비스를 제공할 수 있도록 하였다. 또한 블록체인 서비스 레지스트리 및 컨테이너 레지스트리, 그리고 분산 컨테이너 관리 도구인 쿠버네티스를 이용하여 블록체인 응용서비스의 복제 실행과 실패에 대한 복구를 체계적으로 지원하였다. 더불어 BR2K 기법을 사용할 수 있도록 블록체인 개발 도구인 트러플 프레임워크를 확장하고 이를 구현하여 해당 기법의 실용성을 검증하였다. 추후 BR2K 기법을 이용한 실용성있는 여러 종류의 블록체인 서비스 개발에 대하여 연구할 예정이다.

ACKNOWLEDGEMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education(No. 2019R1I1A3A01052970)

REFERENCES

[1] Seyednima Khezr, et al, "Blockchain Technology in Healthcare: A Comprehensive Review and Directions for Future Research", Applied Sciences, Vol. 9, No. 9, pp. 1736, Apr 2019. DOI: https://doi.org/10.3390/app9091736

[2] M Andoni, et al. "Blockchain technology in the energy sector: A systematic review of challenges and opportunities" Renewable and Sustainable Energy Reviews, Vol. 100, pp. 143-174, Apr 2019. DOI: https://doi.org/10.1016/j.rser.2018.10.014

[3] SN Khan, et al. "Blockchain Technology as a Support Infrastructure in E-Government Evolution at Dubai Economic Department", Proceedings of the international, Jul 2019. DOI: https://doi.org/10.1145/3343147.3343164

[4] Vida J. Morkunas, et al. "How blockchain technologies impact your business model", Business Horizons, Vol. 62, No. 3, pp. 295-306, May 2019. DOI: https://doi.org/10.1016/j.bushor.2019.01.009

[5] MH Kwon, MJ Lee. "A robust execution schemefor Ethereum blockchain application services", Korean Society of Computer Information, Vol. 25, No. 3, pp. 73-80, March 2020. DOI: https://doi.org/10.9708/jksci.2020.25.03.073

[6] MH Kwon, MJ Lee. "Replication of blockchain application services using kubernetes and blockchain service registry", Proceedings of the Korean Society of Computer Information Conference, pp. 363-364, July 2020.

[7] David Balla, et al. "Adaptive scaling of Kubernetes pods", IEEE/IFIP Network Operations and Management Symposium, pp. 20-24s, Aprill 2020. sDOI: 10.1109/NOMS47738.2020.9110428

[8] Truffle framework, *https://www.trufflesuite.com/*

[9] Steve Buchanan, et al. "Container Registries", Introducing Azure Kubernetes Service, pp. 17-34, December 2019. DOI: https://doi.org/10.1007/978-1-4842-5519-3_2

[10] Donguan Huang, et al. "Performance analysis of the raft consensus algorithm", IEEE Transactions on Systems, Man, and Cybernetics: Systems, Vol. 50, No. 1, pp. 171-181, Jan 2020. DOI: 10.1109/TSMC.2019.2895471

[11] Truffle framework, https://etcd.io/

[12] Allison Randal. "The Ideal Versus the Real: Revisiting the History of Virtual Machines and Containers", ACM Computing Surveys, Vol. 53, No. 1, February 2020. DOI: https://doi.org/10.1145/3365199

[13] Bhubaneswar, et al., "An Overview of Smart Contract and Use Cases in Blockchain Technology", 2018 International Conference on Computing, Communication and Networking Technologies (ICCCNT), pp. 10-12, July 2018. DOI:doi:10.1109/ICCCNT.2018.849400

[14] F. Casino, et al., "A systematic literature review of blockchain-based applications: current status, classification and open issues", Telematics and Informatics, Vol. 12, No. 8, pp.55-81, Mar 2019. DOI:10.1016/j.tele.2018.11.006

[15] Brewer, Eric "Kubernetes and the New Cloud", Proceedings of the 2018 International Conference on Management of Data, pp.1, May 2018. DOI:10.1145/3183713.3183725

Authors

Min-Ho Kwon received the B.S/B.A degrees in IT convergence/Economics from University of Ulsan, Korea, in 2020. He is currently an M.S student in Dept. of Electrical/Electronic and Computer Engineering, University of Ulsan. He is interested in blockchain technology, distributed computing, and cloud computing.

Myung-Joon Lee received the B.S. degree in Mathematics from Seoul National University in 1980, and the M.S. and Ph.D. degrees in Computer Science from KAIST in 1982 and 1991, respectively. Dr. Lee joined the faculty of the Department of Computer Science at University of Ulsan, Ulsan, Korea, in 1982. He is currently a Professor in the School of IT Convergence, University of Ulsan. He is interested in blockchain technology, distributed computing, and mobile/cloud service.