공학석사 학위논문

블록체인 응용 서비스 견고성 지원 기법

Techniques for the robustness of blockchain application services

울 산 대 학 교 대 학 원

전기전자컴퓨터공학과
정보통신공학전공

권 민 호

블록체인 응용 서비스 견고성 지원 기법

지 도 교 수 이 명 준

2021년 12월

울 산 대 학 교 대 학 원

전기전자컴퓨터공학과
정보통신공학전공

권 민 호

권민호의 공학석사 학위논문을 인준함

심사위원 이 명 준 (인)

심사위원 윤 석 훈

심사위원 조 동 식

울 산 대 학 교 대 학 원

2021년 12월

[감사의 글]

먼저, 석사 과정 동안 글 쓰는 거 하나 제대로 못 했던 저에게 많은 것을 지도해주신 이명준 교수님께 감사드립니다. 연구실 생활하는 동안 많은 방황을 하고 부족한 모습을 자주 보여줬음에도 교수님은 항상 인내하시면서 다시 저가 스스로 일어나길 기다려주신 좋은 스승이었습니다. 교수님의 지도 덕분에 전문성뿐만 아니라 인생의 내력도 함께 기를 수 있었습니다. 또한, 바쁘신 와중에도 논문 심사를 맡아주신 윤석훈 교수님과 조동식 교수님께도 정말 감사드립니다.

또한, 많은 시간을 연구실에서 함께 보내지는 못했지만, 함께 생활하고 연구했던 모든 선후배님께도 감사의 말씀을 드립니다. 졸업하시기 전까지 항상 귀찮게 질문해도 친절하게 답해주시고 연구실 생활에서 고민도 잘 들어주신 현민 선배, 저가 무뚝뚝하게 굴었음에도 항상 살갑게 다가와 주시고 많은 이야기도 들어주시면서 좋은 조언과 충고도 많이 해주신 여국 선배, 항상 연구실 분위기를 밝게 만들어주고 연구실 사람들을 즐겁게 만들어주는 채린과 연정, 모두 정말 감사했습니다. 마지막으로 저가 선배로서 부족하여 많은 도움을 주지 못해 연구실 생활이 힘들었을 혜원이하고 영은이한테 정말 미안하고, 안 좋은 여건에서도 둘이서 씩씩하게 연구실 생활하는 모습을 보면서 대단하다고 생각해. 남은 기간 별 탈 없이 석사과정 잘 마무리하길 바랄게.

마지막으로, 못난 막내아들을 키우시느라 좋은 옷, 좋은 음식 그리고 귀중한 시간을 희생하신 우리 부모님께 감사합니다. 항상 말도 잘 안 듣고 살갑지도 않은 아들에게, 매일 집에 오면 밥 먹었느냐고 물어봐 주시고 따듯한 밥 차려주시는 우리 어머니, 매일 가족을 위하여 아침 일찍 출근하시고 주말에도 성실하게 일하시면서 힘든 내색 안 하시는 우리 아버지 정말 감사합니다. 저는 항상 부모님에게 바라는 게 많은 못난 아들이었지만, 두 분께서는 항상 조건 없는 사랑을 주셨습니다. 저는 어떠한 누구보다 아버지, 어머니를 존경하고 사랑합니다.

[국문요약]

블록체인 응용 서비스 견고성 지원 기법

블록체인은 중앙 관리자의 개입 없이도 개인과 조직이 정보에 대한 투명한 저장과 관리를 할 수 있다는 기술적 특징 때문에 전 세계의 주목을 받고 있다. 특히, 공공 인프라, 의료, 부동산 그리고 금융과 같이 작업의 히스토리 및 데이터의 신뢰성이 중요한 영역에서 빠르게 개발되어 적용되고 있다. 그러나 블록체인 기반의 서비스가 자연재해, 장비 고장, 사이버 공격과 같은 재난 상황에 의하여 서비스를 제공하지 못하는 경우에, 이를 복구하기 위한 많은 시간과 비용이 필요할 뿐만 아니라, 때로는 회복할 수 없는 손실이 발생할 수 있다. 따라서 블록체인 응용 서비스에 대한 서비스의 연속성 보장 및 신속 재가동을 지원하는 견고성 지원 기법이 필요하다.

본 논문에서는 블록체인 응용 서비스의 견고성을 지원하기 위하여 개발한 BR2K (for Blockchain application, Replication & Recovery technique using Kubernetes) 기법과 활용 도구에 관하여 기술한다. BR2K 기법은 서비스의 복제 실행과 서비스 실패에 대한 신속 재가동을 지원하는 복구 방법을 통해 블록체인 응용 서비스가 사용자에게 지속해서 서비스를 제공하도록 지원한다. 이 기법은 etcd 분산 스토리지를 이용한 사용자 요청의 복제, 사용자 요청 처리의 체계적인 절차, 쿠버네티스 기반의 배포 등을 통해 블록체인 응용 서비스를 복제하여 장애 상황에 대비하고 복제된 서비스들의 일관성을 보장하도록 한다. 또한, 장애 복구 센터 역할을 수행하는 서비스 레지스트리를 블록체인 기반으로 개발하고, 이를 통해 서비스 접속 정보를 사용자에게 언제든지 제공하고 백업 정보를 신뢰성 있게 관리하여 최악에는 블록체인 서비스가 전반적으로 중지되는 상황이 발생하여도 이를 신속히 복구하여 서비스를 제공하도록 한다. 이와 더불어, 제안된 기법의 실용성을 확보하기 위하여 개발된 관련 도구를 설명하고 시범적인 블록체인 응용 서비스에 해당 기법을 적용 및 테스트하여 유효성을 확인한다.

주요어 : 블록체인 서비스, 서비스 견고성, 서비스 복제, 서비스 복원, 서비스 레지스트리

목 차

그 림 목 차

표 목 차

1. 서론

1.1 연구의 배경 및 필요성

분산 장부 기법(Distributed ledger technology)인 블록체인은 기술의 보안성과 신뢰성이 인정되어 부동산, 물류, 에너지, 금융 그리고 공공 서비스 등 다양한 분야에서 빠른 속도로 적용 및 개발되고 있다.[1-5] 특히, 정부의 각종 증명서와 계약, 표결과 같은 디지털화된 기록 및 공공 기관, 금융, 의학, 행정의 인프라 같은 데이터의 신뢰성이 중요한 영역에서는 블록체인 응용 서비스가 더욱 빠르게 개발되고 있다.[6-8] 최근 블록체인 기술의 대표적인 활용으로, 디지털 미디어가 담긴 가상현실 세계인 메타버스와 결합하여 주목받는 블록체인 기반의 디지털 자산 기술인 NFT(Non-Fungible Token)가 있다.[9] 또한, 블록체인을 이용한 인공지능의 학습 데이터에 대한 출처 보장, 학습 데이터에 대한 전처리 과정 및 인공지능의 의사 결정에 대한 투명한 기록 등과 같은 블록체인과 인공지능의 결합은 빠른 속도로 진행되어 점차 실생활에서 블록체인의 역할이 점차 중요해질 것으로 예상된다.[10-11]

그러나 블록체인 기술이 빠른 속도로 발전하고 있지만, 실제 기업 환경에서 바로 활용하기에는 아직 부족한 점이 많으며, 블록체인이 가진 기술적 복잡성에 의하여 많은 기업이 상용 서비스 개발에 어려움을 겪고 있다. 그에 반해, 메이저 클라우드 서비스 기업인 IBM, 마이크로소프트, 오라클 등이 클라우드 환경에서 블록체인의 구성과 블록체인 어플리케이션 개발 도구를 지원하는 서비스를 선보이고 있으며 구글과 아마존도 이더리움 블록체인, 하이퍼레저 블록체인과 연결된 클라우드 서비스를 제공하고 있다. 클라우드를 바탕으로 한 블록체인 응용 서비스는 개별 노드보다는 클라우드의 특성상 가용성과 견고성을 어느 정도 지원하지만, 개별 기업의 서비스에 대한 의존은 2020년 11월 아마존 클라우드 서비스의 대규모 접속 장애와 같은 위협에 근본적으로 노출될 수밖에 없으며 서비스와 관련된 민감한 데이터 유출의 한계를 벗어날 수 없다.

또한, 대규모 인프라나 클라우드, 의료, 금융과 관련된 블록체인의 응용 서비스가 각종 재난과 해킹과 같은 상황에 의해 장애나 파괴로 서비스가 실패할 경우 그 부작용을 예측하기 어려우며 복구하기 위한 막대한 비용이 발생할 수 있다. 이에 반해, 현재 블록체인 응용 서비스의 견고성에 대한 체계적인 연구는 활발히 이루어지고 있지 않은 상황이다. 따라서 본 연구에서는 블록체인 기반의 응용 서비스를 복제하여 장애 상황에서도 서비스의 연속성을 확보하고 다수의 실행 노드 파괴와 같은 재난 상황을 극복하기 위한 신속 재가동 기법을 제안한다.

1.2 연구의 개요

본 논문은 블록체인 응용 서비스의 견고성을 지원하는 기법에 대해 제안한다. 이 견고성 지원 기법에는 블록체인 응용 서비스가 장애나 결함에도 지속적인 서비스를 제공하기 위하여 서비스를 일관성 있게 복제하는 방법, 이 방법을 적용한 블록체인 응용 서비스를 분산 환경에서 신속하게 배포하기 위한 방안 그리고 블록체인 응용 서비스가 재난 상황에 의하여 전면적으로 실패하였을 경우 체계적인 복구를 지원하는 방법이 포함된다. 본 논문에서는 제안하는 블록체인 응용 서비스의 견고성을 지원하는 기법을 BR2K(for Blockchain application, Replication & Recovery technique using Kubernetes)이라고 명명한다.

BR2K 기법은 Raft(Reliable, Replicated, Redundant and Fault-Tolerant) 컨센서스[12] 프로토콜을 사용하는 분산 저장소인 ETCD(/etc, distributed)[13] 기반으로 개발된 서비스 복제 방법을 지원하여, 어떠한 경우에서도 일관성을 보장하면서 블록체인 응용 서비스의 연속성을 지원할 수 있다. 또한, 이 기법에서는 가상화 기술인 컨테이너와 분산 컨네이너 관리 도구인 쿠버네티스를[14] 이용하여, BR2K 기법이 적용된 블록체인 응용 서비스의 신속하고 일관된 배포를 지원한다. 더불어, 이와 같이 배포된 서비스 접근 위치를 제공하고 서비스 복구 정보에 대한 저장 및 관리 그리고 서비스 이용자의 복구 요청 기능 등을 제공하는 서비스 레지스트리를 이더리움 블록체인의 스마트 컨트랙트[15-16] 기반으로 개발하여, 본 기법에서는 재난 사태에 의한 블록체인 응용 서비스의 전반적인 실패에도 체계적이고 신속한 복구를 지원할 수 있다.

본 논문에서는 제안한 BR2K 기법을 실제 개발 환경에서 손쉽게 사용하기 위한 프레임워크와 웹 브라우저 환경에서 동작하는 모니터링 도구를 개발한다. 이 프레임워크는 실제 개발 환경에서 많이 사용되는 프레임워크들을 기반으로 확장하고 개발하여, 해당 기법을 실용적인 블록체인 응용 서비스에 손쉽게 적용할 수 있도록 한다. 또한, 개발할 모니터링 도구는 웹 브라우저 환경에서 BR2K 기법을 적용한 블록체인 응용 서비스에 대한 서비스 상태를 지속적으로 확인하고 결함 복구를 지원할 수 있도록 한다. 이와 더불어, 본 기법의 유용성을 확인하기 위하여, 시범적인 블록체인 응용 서비스에 BR2K 기법을 적용하고 테스트한다.

1.3 논문의 구성

논문의 구성은 다음과 같다. 다음 장에서는 배경지식으로 분산 컨센서스 기법인 Raft와 이를 기반으로 동작하는 분산 저장소인 ETCD, 블록체인 플랫폼인 이더리움과 스마트 컨트랙트 그리고 분산 컨테이너 관리 도구인 쿠버네티스에 대해서 다룬다. 또한 3장에서는 블록체인 응용 서비스의 견고성을 지원하는 BR2K 기법에서 서비스를 복제하는 방법에 관하여 기술하고, 4장에서는 BR2K 기법이 적용된 블록체인 서비스를 신속하고 일관된 배포를 지원하는 방법과 해당 서비스의 실패에 대한 신속하고 체계적인 복구 방법을 설명한다. 이와 더불어, 5장에서는 BR2K 기법을 실제 개발 환경에서 손쉽게 적용하기 위한 도구들의 개발에 대해서 설명하고 6장에서는 실용적인 블록체인 응용 서비스에 BR2K 기법을 적용하고 이를 테스트하여 본 기법의 유효성에 대하여 논한다. 마지막으로 7장에서는 결론을 제시한다.

2. 관련 연구

본 장에서는 대중적인 컨센서스 알고리즘인 Raft의 개념과 이를 기반으로 동작하는 분산 키-값 저장소인 ETCD에 대하여 기술한다. 또한, 2세대 블록체인인 이더리움과 스마트 컨트랙트에 대하여 설명하고 가상화 기술인 컨테이너 및 분산 컨테이너 관리 도구인 쿠버네티스도(Kubernetes) 소개한다.

2.1 Raft 컨센서스와 ETCD

Raft는 분산 환경에서 참여 노드들이 충돌 오류와 같은 상황에도 복제 로그를 관리하기 위한 합의 알고리즘이다. Raft가 등장하기 이전에는 분산 환경에서 발생하는 합의 문제를 Paxos[17-18] 합의 알고리즘으로 해결하였다. 그러나 Paxos는 합의 문제를 해결하는 과정이 길고 이해하기 힘든 단점이 존재한다. Paxos와 같은 성능을 내면서도 합의 문제를 이해하기 쉽게 해결하기 위하여 Raft가 개발되었다. Raft는 합의 그룹의 참여 노드 중 과반수에 장애가 발생하기 전까지 합의 그룹들은 정상적으로 동작함을 보장하며 분산 시스템의 검증 도구인 Verdi를 통해 유효성을 검증받았다. 또한, Raft는 여러 산업에 사용되고 있는 블록체인 플랫폼인 하이퍼레저, 비즈니스를 위한 블록체인인 아르고, 신뢰성 있는 분산 저장소인 ETCD등 다양한 개발 환경에서 사용되고 있다. Raft는 분산 그룹의 참여 노드들이 발생하는 합의 문제를 이해하기 쉽게 해결하기 위하여 강력한 리더, 로그 관리 및 안정성인 3가지로 분리하였다. Raft에서 참여 노드의 상태는 리더, 팔로워, 후보자인 3가지 상태가 있으며 상태 변이를

통해 분산 합의 그룹 간의 안정성을 유지한다. 또한 참여 노드들은 Raft 알고리즘에 의해 강력한 리더를 선출하여 분산 환경에서 동일한 상태를 가질 수 있도록 로그를 복사 및 관리 작업을 수행하도록 한다.

ETCD는 분산 시스템 또는 시스템 클러스터에서 데이터를 안정적으로 저장할 수 있는 Raft 기반의 키-값 분산 저장소이다. 많은 개발 환경에서 ETCD는 간단한 웹 응용 서비스뿐만 아니라 도커[19] 등의 컨테이너에 대한 오케스트레이션 도구인 쿠버네티스 같은 복잡한 응용 서비스에서도 사용되는 신뢰성 있는 분산 저장소이다. ETCD는 분산 환경을 구성하는 각 노드에서 복제되어 실행되며, Raft 컨센서스 알고리즘에 의하여 하나의 리더 상태를 가진 ETCD와 다수의 팔로워 상태를 가진 ETCD들로 구성된다. 리더 상태인 ETCD에 의하여 일관성을 보장하는 형태로 로그가 복제되고 관리가 된다. ETCD는 데이터의 저장 기능뿐만 아니라 저장된 데이터 관찰 및 변화에 대한 이벤트 기능을 지원하여 저장된 데이터를 지속적으로 관리하는 데도 용이하다. 또한 ETCD는 조건에 따라 여러 연산을 수행할 수 있는 조건 트랜잭션 기능, 분산 그룹 내에서 발생할 수 있는 실패에 대한 일관성 있는 정책, 복구 기능 그리고 사용자 인증서를 통한 자동 TLS(Transport Layer Security)를 지원하여 더욱 안전한 실행을 보장한다.

2.2 이더리움 블록체인과 스마트 컨트랙트

블록체인은 분산 컴퓨팅 기술 기반의 데이터 위변조 방지 기술이다. P2P 방식을 기반으로 사용자의 요청에 해당하는 트랜잭션을 블록이라는 단위 데이터에 저장하며, 블록을 체인 형태로 연결하여 합의 알고리즘을 통해 분산 환경을 구성하는 노드에 복제한다. 이러한 특성 때문에 블록체인에 저장된 데이터는 임의로 수정할 수 없으며 변경의 결과를 누구나 열람할 수 있다. 이더리움은 2세대 블록체인으로 스마트 계약 기능을 제공하기 위하여 고안된 블록체인 플랫폼이다. 이더리움 블록체인은 1세대 블록체인에 없는 반복문과 제어문을 사용할 수 있는 튜링 완전성을 가지고 있어 간단한 거래뿐만 아니라 복잡한 형태의 탈중앙화 응용 서비스를 만들 수 있는 장점이 있다. 이더리움은 p2p 네트워크, 전자서명, 암호 해시 등 기본적인 기능을 제공한다. 또한 사용자에 의해 생성된 트랜잭션과 트랜잭션에 관련된 데이터들이 모여 있는 블록의 유효성을 검증하는 합의 엔진 등을 관리한다. 이더리움은 스마트 컨트랙트의 실행을 이더리움 가상 머신을 통하여 처리하며 스마트 컨트랙트의 각종 데이터 구조를 정의하고 관련된 데이터를 관리한다.

스마트 컨트랙트란 실제 사람이 대면하는 기존 계약 방식에서 벗어나 블록체인상에서 조건에 따라 자동으로 수행하는 스마트 계약 기능이다. 스마트 컨트랙트는 이더리움 블록체인 기반에서 실행되기 때문에 무결성 보장 및 조작 방지가 가능한 실행 코드이다. 계약을 원하는 사용자는 배포된 스마트 컨트랙트를 이용하기 위해 트랜잭션을 보내고 스마트 컨트랙트에 정의된 내용에 따라 기능을 받는다. 또한 스마트 컨트랙트에서는 기능을 수행하기 전후로 특정한 기능을 수행할 수 있도록 하는 함수 변경자, 트랜잭션 내에서 호출되어 특정 값을 클라이언트에 전달하고 트랜잭션에 로그 형태로 저장되는 이벤트 기능을 통해 스마트 컨트랙트의 기능은 안전하고 다양한 형태로 사용될 수 있다.

2.3 컨테이너와 쿠버네티스

컨테이너(Container)는 기존의 가상화 기술의 단점을 극복한 경량 가상화 기술이다. 컨테이너 기술은 임의의 환경에서도 제약 없이 신속하게 실행하기 위하여, 응용 서비스 실행에 필요한 모든 정보를 컨테이너 이미지라는 파일 형태로 빌드하고 이 이미지를 기반으로 응용 서비스를 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 한 번에 배포할 수 있다. 쿠버네티스에서 컨테이너로 구성된 파드가 일반적인 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. BR2K 기법의 서비스 복제

BR2K는 블록체인 응용 서비스의 견고성을 보장하기 위하여, 서비스를 복제하여 실행하고 복제 서비스의 전반적인 실패에도 신속하고 체계적인 복구를 지원하는 기법이다. 이 기법은 블록체인 응용 서비스를 분산 환경에서 복제하여, 복제된 서비스가 일부 중지되어도 사용자에게 지속해서 서비스를 제공할 수 있다. 본 장에서는 블록체인 응용 서비스가 장애 상황에서도 신뢰할 수 있는 형태로 서비스를 제공하도록 하는 서비스 복제 방법에 대하여 자세히 기술한다.

3.1 BR2K 서비스

블록체인 응용 서비스의 복제는 독립적인 노드들로 구성된 분산 환경에서 블록체인 응용 서비스를 복제하여 실행하고 각 응용 서비스의 상태를 동기화하는 것이다. 그러나 분산 환경에서 독립적으로 실행된 각 응용 서비스들의 상태를 어떠한 상황에서도 일관성을 보장하면서 동기화하기는 쉽지 않다. 본 절에서는 일반적인 블록체인 응용 서비스의 구조와 이와 같은 서비스를 복제하는 일반적인 방법에 대하여 논한다. 또한, 이러한 복제에 관한 문제점과 이를 극복하기 위한 BR2K 기법에서 블록체인 응용 서비스를 복제하는 방법에 대해서 상세히 설명한다.

3.1.1 일반적인 블록체인 응용 서비스의 구조 및 복제

블록체인 응용 서비스는 일반적으로 웹 기반으로 동작하는 응용 서비스이며 3가지 형태로 구분된다. 첫째는 응용서비스 실행에 필요한 애플리케이션 특정 데이터가 모두가 블록체인에 기록된 정보로서 존재하거나 사용자의 요청 역시 블록체인의 트랜잭션 수행만으로 필요한 정보 변경이 이루어지는 경우이다. 둘째는 응용서비스가 블록체인에 저장된 데이터 외에 자신의 로컬 스토리지, 로컬 데이터베이스에 서비스 운영에 필요한 정보를 저장하는 방법이다. 셋째는 응용서비스가 블록체인에 저장된 데이터 외에 네트워크에 분산되어 운용되는 분산 스토리지나 분산 데이터베이스에 서비스 운영에 필요한 정보를 저장하는 것이다. 그림 1은 위와 같이 3가지 형태로 분류된 서비스를 결합하여 일반화시킨 블록체인 응용서비스의 구조이다.

블록체인 응용 서비스는 서비스가 중지되어도 유지되어야 하는 정보들의 집합인 상태를 가지고 있다. 이러한 상태는 그림 1과 같이 서버가 실행되는 노드에 존재하는 로컬 상태(Local state)와 서버가 실행하는 노드 외부에 존재하는 외부 데이터베이스의 상태, 블록체인에 기록되는 정보 등을 포함하는 외부 상태(External state)로 구분된다. 특히, 로컬 상태는 응용 서비스의 서버의 런타임 변수 데이터나 파일의 형태로 서버의 실행 노드에 저장되는 서비스 메타 정보, 로컬 데이터베이스의 상태 등의 정보가 포함된다.

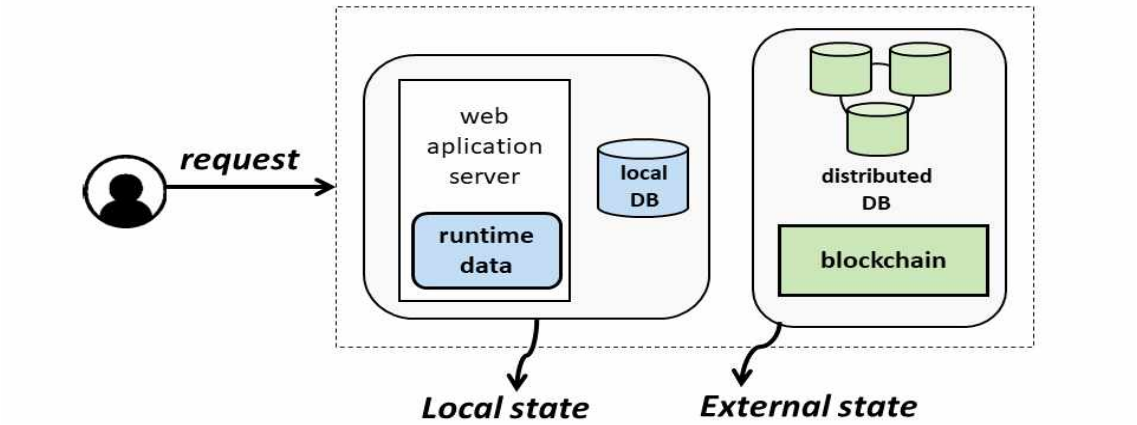

그림 1. 블록체인 응용 서비스의 구조

이와 같은 블록체인 응용 서비스의 복제는 분산 환경을 구성하는 각 노드에 블록체인 응용 서비스의 서버를 중복하여 실행한다. 복제 실행된 각 서버는 그림 2와 같이 자신이 실행되고 있는 노드에서 유지하지 않는 외부 데이터베이스의 상태, 블록체인에 저장되는 정보 등을 포함하는 하나의 외부 상태를 가지며 모든 서버가 공유하는 상태이다. 또한, 각 블록체인 응용 서비스의 서버는 자신의 실행 노드에만 있는 고유한 로컬 상태를 가지며, 이 상태들은 각 서버끼리 지속적으로 통신하여 동기화된다.

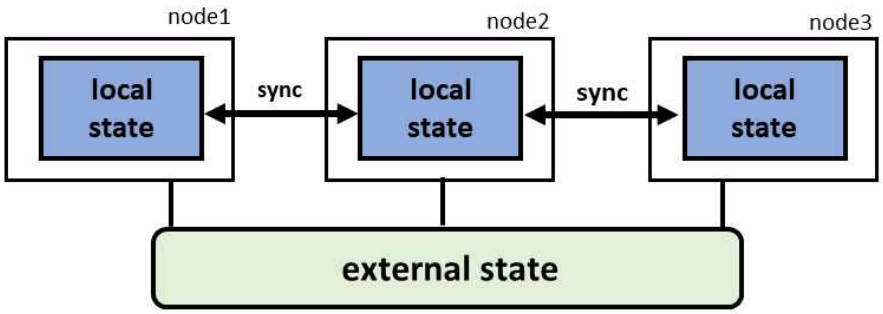

그림 2. 복제된 블록체인 응용 서비스의 상태

복제된 블록체인 응용 서비스에서 각 서버 실행 노드에 존재하는 독립적인 로컬 상태는 장애 상황이 발생하여도 같아야 하므로, 로컬 상태에 대한 동기화 작업은 Raft와 같이 검증된 컨센서스 프로토콜을 기반으로 이루어진다. 그림 3은 복제된 블록체인 응용 서비스의 사용자 요청에 대한 처리를 통한 상태 동기화 과정을 나타낸다. 사용자 요청과 같이 서비스의 상태를 변경하는 이벤트가 한 서버에 발생했을 때, 해당 서버는 컨센서스 모듈을 이용하여 서버들에게 해당 이벤트에 대한 정보를 복제하고, 각 서버는 복제된 정보를 처리하여 자신의 로컬 상태를 변경하여 동기화한다.

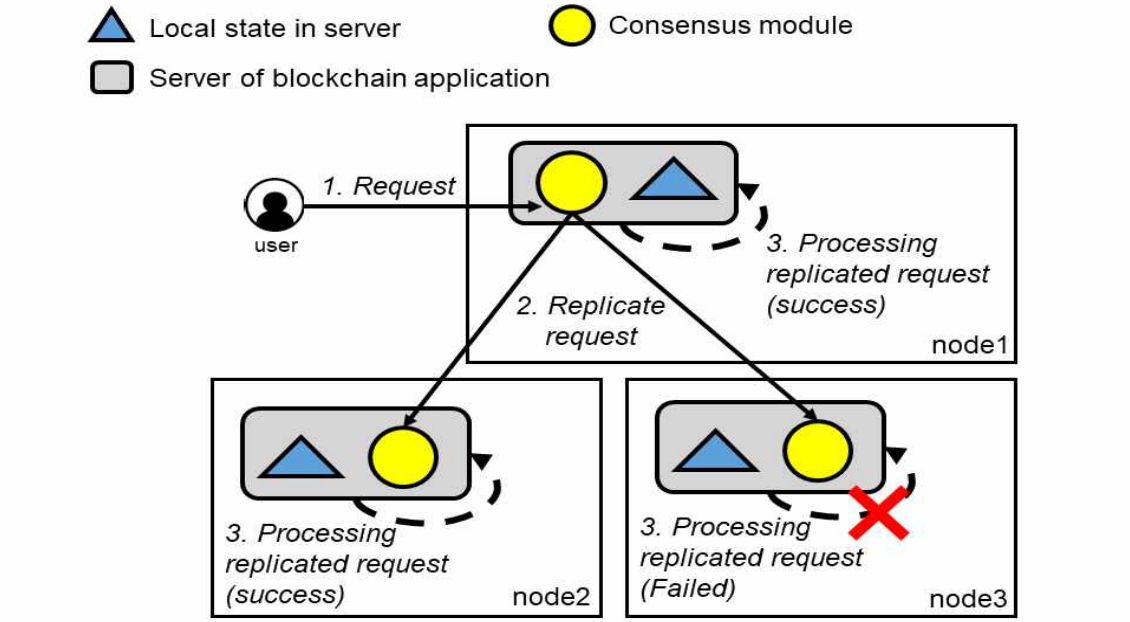

그림 3. 복제된 블록체인 응용 서비스의 사용자 요청 처리 과정

그러나, 이와 같은 방식은 서비스의 로컬 상태에 대한 일관성을 보장하지 못한다. 그림 3에 3번 노드의 서버처럼 일부 서버가 복제된 사용자 요청을 처리하는 것에 실패하거나 일부만 성공한다면, 각 서버의 로컬 상태는 하나의 상태로 동기화되지 못하며 이 상태를 롤백하는 작업도 쉽지 않다. 또한, 해당 사용자 요청이 로컬 상태가 아닌 외부 데이터베이스나 블록체인 노드에 유지되는 외부 상태를 변경하는 요청이라면, 각 서버가 공유하는 외부 상태를 중복하여 변경하는 문제가 발생할 수 있다. 더불어, 해당 방식은 서비스 사용자가 분산 환경에서 구성되는 서버 중에 어떤 서버에 접근하여 서비스를 요청해야 하는지에 대한 서비스 디스커버리 문제, 사용자 요청과 같은 정보를 끊임없이 복제하여 각 서버의 로컬 노드에 대한 용량이 초과하는 문제 등이 발생 할 수 있다.

3.1.2 BR2K 서비스의 구조

BR2K 기법에서는 3.1.1에서 설명한 문제점을 극복하기 위하여 상태를 변경하는 사용자 요청과 중복 실행된 블록체인 응용 서비스의 서버에 대한 역할을 체계적으로 분류하고, 각 서버들 간의 일관성을 보장하는 형태로 복제하기 위하여 Raft 컨센서스 프로토콜 기반의 분산 저장소인 ETCD를 이용한다.

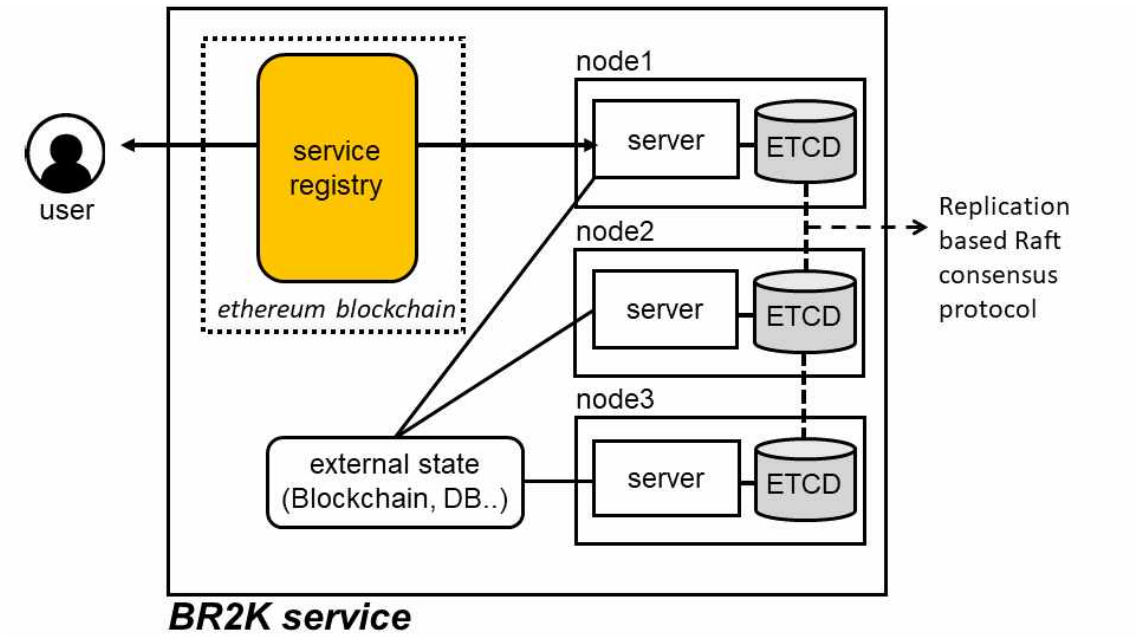

그림 4. BR2K 서비스 구조

본 논문에서는 BR2K 기법을 적용하여 복제된 블록체인 응용 서비스를 BR2K 서비스로 명명한다. 그림 4는 BR2K 서비스의 구조를 나타낸다. BR2K 서비스에서는 분산 환경을 구성하는 각 노드에 블록체인 응용 서비스에 대한 서버와 사용자 요청을 복제하기 위한 ETCD가 같이 실행된다. 또한 BR2K 서비스에서는 이더리움 블록체인 기반으로 서비스 접속 정보를 사용자에게 제공하고 복구 작업을 위한 백업 메타 정보를 관리하여 장애 복구 센터 역할을 수행하는 서비스 레지스트리도 있다.

BR2K 서비스를 구성하는 블록체인 응용 서비스의 서버 역할에는 리더(Leader), 팔로워(Follwer), 레지스터(Register), 러너(Learner)가 있으며, BR2K 서비스는 일반적인 상황에서는 하나의 리더와 다수의 팔로워로 구성된다. 리더 역할을 수행하는 응용 서비스의 서버는 사용자에게 서비스를 제공하고 서비스의 최신 상태를 업데이트한다. 팔로워 서버는 리더가 업데이트한 최신 서비스 상태를 주기적으로 따라가는 역할이

다. 레지스터 서버는 자신이 새로운 리더로서 역할을 수행하기 전에, 팔로워 역할에서 업데이트하지 못한 최신 서비스 상태를 동기화하고 서비스 레지스트리에 자신의 위치를 새로운 서비스 제공 위치로 업데이트하는 역할이다. 러너는 한 서버가 상태에 대한 일관성을 보장하지 못할 때, 서비스 레지스트리를 이용하여 백업된 안정적인 상태를 가져오는 작업을 수행하는 역할이다.

BR2K 서비스에 있는 각 역할은 특정 작업을 수행하기 위한 일련의 과정 절차를 모은 액션(Action)이라는 단위로 작업을 수행한다. 그림 5는 BR2K 서비스의 각 역할에 대한 액션 및 전이 과정을 나타낸다. BR2K 서비스가 시작하면, 실행된 각 응용 서비스의 서버는 팔로워로 시작한다. 실행된 모든 서버는 리더 역할을 수행할 수 있는 조건인 Leader condition을 주기적으로 검사하는 *Check role* 액션이 수행되며 이 액션은 역할이 전환되어도 계속 수행된다. 이 액션에서 Leader condition은 각 응용 서비스의 서버와 연결된 ETCD의 상태가 ETCD 내부 로그 관리하는 대표자라면 TRUE가 되고, 아니라면 FALSE가 된다. Leader condition이 TRUE인 팔로워 서버는 레지스터 서버 역할로 전환되어 서비스 준비 과정을 거친 뒤, 사용자에게 서비스를 제공하는 리더가 된다.

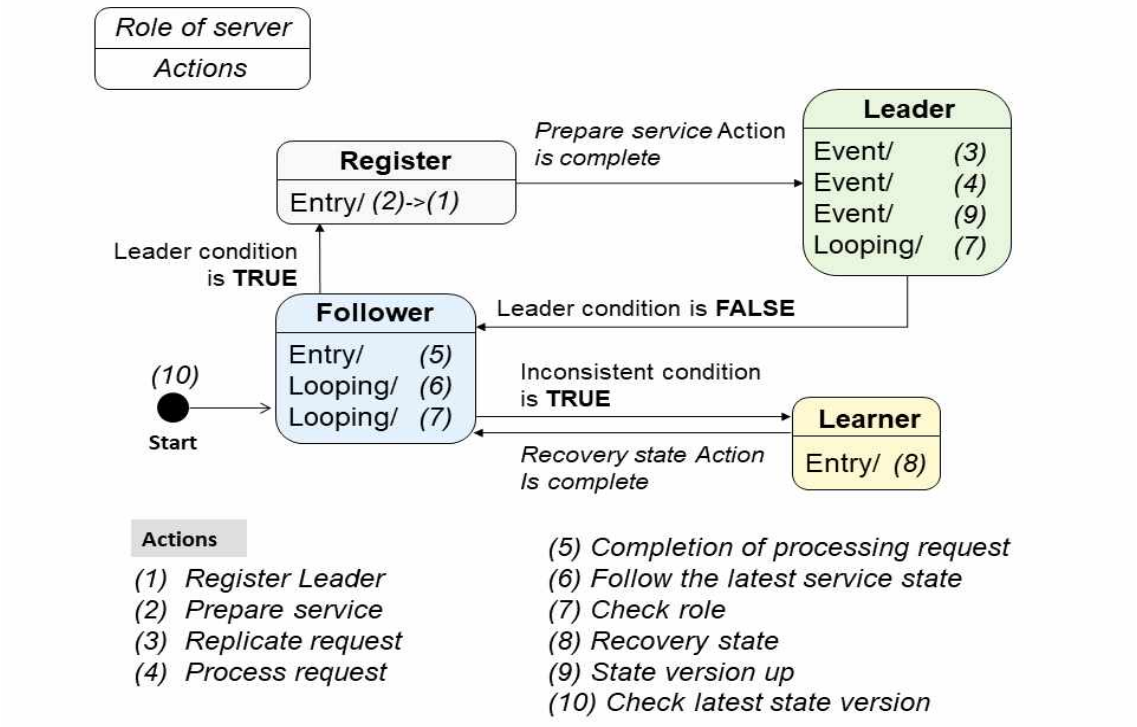

그림 5. BR2K 서비스의 서버 역할 및 액션

BR2K 서비스에서는 복제된 블록체인 응용 서비스의 상태를 체계적으로 동기화하기 위하여, 사용자 요청을 표 1과 같이 Replication, Only-once, External-txn인 3가지 타입으로 분류한다. Replication 타입은 서버의 로컬 상태를 변경하는 사용자 요청으로, 모든 서버가 처리해야 하는 사용자 요청이다. Only-once 타입은 읽기 작업, 외부 상태를 변경하는 작업과 같이 오직 한 번만 실행해야 하는 사용자 요청이며 오직 리더에 의해서만 처리된다. External_txn 타입은 사용자 단에서 발생하여 성공한 블록체인 트랜잭션에 대한 로그를 서버 단에서 복제하여 관리하기 위한 사용자 요청이다. 이 타입은 리더에 의해서 복제만 되고 모든 서버가 처리하지 않는 요청이다.

표 1. 분류된 사용자 요청 및 ETCD에 저장되는 데이터

사용자 요청	
종류	**설명**
Replication	모든 서버가 처리해야 하는 사용자 요청
Only-once	읽기 작업, 외부 데이터베이스 트랜잭션을 실행시키는 작업과 같이 오직 리더 서버만 처리하는 사용자 요청
External-txn	사용자 단에서 성공한 블록체인 트랜잭션에 대한 로그 정보가 담긴 사용자 요청
ETCD에 저장되는 데이터	
종류	**설명**
RLE(Request Log Entry)	인덱스 형태로 저장된 사용자 요청 (key, value) : (인덱스, 사용자 요청)
RS(RLE State)	사용자 요청에 대한 종류, 상태 그리고 저장한 주체를 나타내는 메타 데이터 (key, value) : (“RS”_인덱스, 사용자 요청의 종류, 타입, 저장한 주체) 사용자 요청의 타입: Null, InPorcess, Success, Failed, Ignore
LS(Log Size)	BR2K 서비스에서 저장된 사용자 요청의 총 개수 (key, value) : (“Log_Size”, 사용자 요청 로그의 총 개수)
SSI(Service State Index)	BR2K 서비스의 최신 상태를 나타내는 인덱스 (key, value) : (“Service_State”, 최근에 처리된 사용자 요청 로그의 인덱스)
LPI(Latest Processed Index)	각 서버가 마지막으로 처리한 사용자 요청에 대한 인덱스 (key, value) : (서버 ID, 서버가 마지막으로 처리한 사용자 요청의 인덱스)
SV(State Version)	BR2K 서비스의 최신 상태 버전 (key, value) : (“Service_State_Version”, 현재 BR2K 서비스의 상태 버전)
SSV(Server State Version)	각 서버가 유지하고 있는 상태에 대한 버전 (key, value) : (서버 ID_"State_Version", 현재 서버의 상태 버전)
CL(Current Leader)	최신 리더 서버의 아이디 (key, value) : ("Current_Leader", 현재 리더 서버의 ID)

BR2K 서비스는 서비스를 제공하는 리더 서버에 대한 중지와 같은 상황에서도 일관성을 보장하면서 서비스의 상태를 동기화하기 위하여, 표 1과 같이 ETCD에 사용자 요청 이외에도 여러 종류의 메타 데이터들도 저장하여 복제한다.

3.1.3 BR2K 서비스의 동작 과정

BR2K 서비스의 복제는 사용자가 서비스를 받기 위하여 리더에게 사용자 요청을 보낼 때 시작한다. 사용자 요청을 받은 리더는 해당 요청을 바로 처리하지 않고 ETCD에 저장하여 모든 서버에 복사하는 작업인 *Replicate Request* 액션을(§fig. 5, 6) 수행한다. 하나의 사용자 요청에 대한 Replicate Request 액션의 작업 절차는 다음과 같다.

[3.1 in fig. 6] 모든 사용자 요청에 대한 복제 실행을 피하고자, 사용자 요청을 Replication, Only-once, External-txn와 같이 3가지 형태로 (§table. 1) RESTful API 기반으로 분류한다.

[3.2 in fig. 6] 리더는 분류된 요청이 BR2K 서비스에서 몇 번째 요청인지 나타내는 인덱스 값을 할당한다. 인덱스 값은 현재 사용자 요청의 총 개수를 나타내는 *LS* (Log Size, §table.1)의 값에 1 증가 시켜 할당한다. 요청에 대한 인덱스 할당이 완료되면, 해당 요청을 인덱스 형태의 로그인 *RLE* (Request Log Entry, §table.1)로 ETCD에 저장한다. 또한 해당 요청의 처리 과정에서 발생할 수 있는 장애나 실패를 극복하기 위하여, 해당 요청에 대한 종류, 처리 상태 그리고 이 요청을 저장한 리더 서버의 ID에 대한 메타 데이터를 *RS* (RLE State, §table.1) 형태로 ETCD에 저장한다. 초기 RS의 처리 상태 값은 *Null* (§table.1)로 저장된다. 더불어 저장된 요청의 개수가 ETCD 내에서 증가하기 때문에, *LS* (Log Size, §table.1)의 값도 1 증가 시켜 업데이트한다.

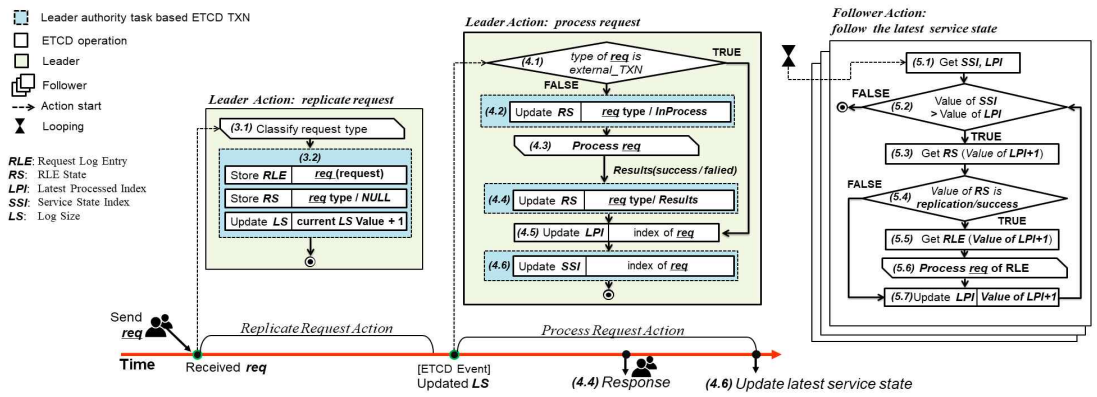

그림 6. BR2K 서비스의 사용자 요청 처리 과정

Replicate request 액션으로 인하여 ETCD에서 LS의 업데이트가 완료되면, ETCD 내에서 LS 업데이트 이벤트가 발생한다. 이는 ETCD에 의해, 업데이트된 LS의 값을 인덱스로 하는 사용자 요청이 BR2K 서비스를 구성하는 다수 서버에게 복제되었음을 의미한다. 이 이벤트를 감지한 리더 서버는 해당 요청을 처리하고 응답하는 작업인 *Process Request* 액션을(§fig. 5, 6) 수행한다. 이 액션의 과정은 다음과 같다.

[4.1 in 6] 리더가 처리해야 하는 사용자 요청의 종류가 처리 작업이 필요 없는 *External_TXN* (§table.1) 타입인지 확인한다. 이 요청의 종류가 External_TXN가 아니라면, 리더 서버는 이 요청에 대한 처리 작업을 수행한다.

[4.2-4 in 6] 리더는 해당 요청의 처리 중에 발생할 수 있는 서버의 중지나 역할 전환에 대응하기 위하여, 해당 요청에 대한 RS의 처리 상태를 *InProcess* (§table. 1) 상태로 업데이트한다. RS의 상태에 대한 업데이트가 완료되면, 리더는 해당 요청에 대한 처리 작업을 수행하고 처리 결과에 대한 응답을 사용자에게 보낸다. 리더는 요청에 대한 처리 결과를 *Success, Fail* (§table.1)로 구분하며, 이 결과를 해당 요청에 대한 RS의 처리 상태 값으로 업데이트한다. 이 작업으로 인하여 그림 6의 (5.4)처럼 팔로워들은 처리 결과가 Fail이 아니고 타입이 *Replication* 인 요청에 대해서만 처리 작업을 수행하여 불필요한 요청의 처리를 피하게 된다.

[4.5-6 in 6] 마지막으로 리더는 자신이 복제된 사용자 요청을 어디까지 처리하였는지를 나타내는 *LPI* (§table.1, Latest Processed Index)를 현재 처리한 요청의 인덱스로 업데이트한다. 또한, 리더는 BR2K 서비스의 최신 서비스 상태를 인덱스로 형태로 나타내는 *SSI* (§table.1, Service State Index)도 처리한 요청의 인덱스로 업데이트한다.

그림 6에 LS, SSI, RS의 업데이트와 RLE의 저장 작업은 리더만 작업할 수 있는 리더 권한 작업(Leader authority task, §fig. 6)이다. BR2K 서비스의 서버가 리더 권한을 가지기 위해서는 Leader condition이 TRUE이고 현재 리더를 나타내는 *CL*(Current Leader, §table. 1)의 값을 리더가 될 서버가 가지고 있는 고유한 ID 값으로 업데이트해야 한다. 리더 권한 작업은 ETCD에서 제공하는 기능인 조건 트랜잭션 기반으로 동작하여, ETCD가 리더 권한 작업을 수행하려고 하는 서버에 대해서 리더 권한을 검사한다. 만약, 이전 리더가 새로운 리더가 등장했음에도 리더 권한 작업을 수행하려고 한다면, ETCD에 의하여 해당 작업은 거절되고 이전 리더의 역할은 팔로워로 전환되어 ETCD에 저장되는 데이터들이 중복해서 변경되는 일을 방지한다.

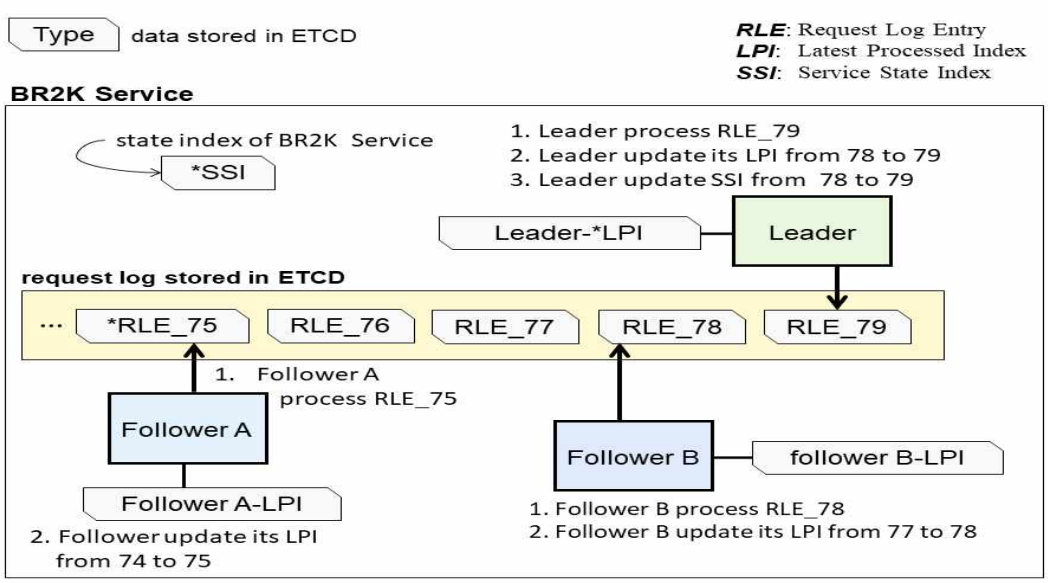

그림 7. BR2K 서비스의 LPI 업데이트 과정

리더가 아닌 팔로워 역할을 가진 서버들은 리더로 인하여 변경된 BR2K 서비스의 최신 상태를 따라가는 *the latest service state* 액션을(§fig. 5, 6) 주기적으로 수행한다. 이 액션에서는 각 팔로워가 자신이 마지막으로 처리한 사용자 요청에 대한 인덱스 값인 LPI와 최신 서비스 상태를 나타내는 SSI와 같은지를 비교하여, BR2K 서비스가 최근에 서비스 상태를 업데이트하였는지 확인한다. 팔로워는 SSI와 자신의 LPI 값이 서로 같지 않다면, 최신 상태가 변경되었다고 판단한다. 그리고 팔로워 자신이 처리하지 못한 사용자 요청 로그들을 ETCD에서 하나씩 가져와 처리하고 그림 7과 같이 자신의 LPI의 값을 1 증가시킨다. 이때, 각 팔로워는 가져온 사용자 요청 중에 사용자 요청의 타입인 Replication이고 리더에 의하여 처리에 성공한 사용자 요청만을 처리하여 자신의 로컬 상태를 업데이트한다.

3.2 서비스 상태에 대한 일관성 보장

BR2K 서비스는 사용자에게 서비스를 제공하는 하나의 리더와 리더의 중지를 대비한 복제본 서버인 팔로워들로 구성된다. 본 절에서 사용자에게 서비스를 제공하는 리더 서버가 중지되었을 때, BR2K 서비스의 상태에 대한 일관성을 보장하면서 서비스를 지속하는 것에 대하여 자세히 기술한다.

3.2.1 새로운 리더

BR2K 서비스는 예상하지 못한 런타임 버그, 하드웨어 부족, 정전 그리고 네트워크 파티션과 같은 이유로 서비스를 제공하는 리더와 일부 팔로워가 중지되어 서비스를 제공하지 못할 수 있다. 이와 같은 장애 상황이 발생하면, BR2K 서비스에서 실행되고 있는 과반수의 팔로워 서버들은 새로운 서비스 제공자인 리더를 새롭게 선출해야 한다.

표 2. Prepare service 액션 과정

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. Prepare Service
 1:   lpi ← GET(serverID of server)                    /*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 type of rs is replication & state of rs is success
 9:           request ← GET(curIndex)
10:           result ← PROCESS request
11:           UPDATE('RS/'+curIndex, result)
12:       UPDATE('Latest_Processed_Index', curIndex)
13:   UPDATE('Service_State_Index', logSize)
14:   WATCH('Log_Size')
```

장애 상황에 의하여 이전 리더가 중지되면 BR2K 서비스에서는 각 팔로워들은 주기적으로 실행되는 Check role 액션에 의하여 새로운 리더가 될 팔로워가 선출된다. 이때, 해당 팔로워는 바로 리더 역할로 전환되지 않고 레지스터 역할 거쳐 새로운 리더가 된다. 먼저, 레지스터 서버는 자신의 상태가 마지막으로 업데이트된 최신 서비스 상태가 되기 위하여 *Prepare service* 액션을(§fig. 5) 수행한다. 해당 액션에서는 이전 리더에 의해서 불필요한 요청의 처리를 피하기 위하여, 복제된 요청을 처리 상태와 타입에 따라 구분지어 처리한다.

[line 6-7 in table.2] 레지스터 서버는 이전 리더가 복제만 하고 처리하지 못한 Null 상태인 사용자 요청을 만나게 된다면 자신이 이 요청의 처리 결과에 대해 응답을 할 수 없다고 판단하여, 해당 요청의 RS에 대한 상태 값을 처리 불가 상태 값인 *Ignore*로 업데이트한다.

[line 10-12 in table.2] 레지스터 서버는 오직 요청 타입이 Replication이고 처리 상태가 Success인 요청에 대해서만 처리하고 이에 대한 처리 결과를 해당 요청의 처리 상태에(processing of RS) 업데이트한다. 처리하는 도중에 중지된 상태를 나타내는 InProcess 상태를 가진 사용자 요청은 레지스터가 이 사용자 요청에 대하여 어떠한 부분까지 처리되었는지를 파악할 수가 없어, 리더가 될 레지스트리 서버는 이 처리 상태를 가진 사용자 요청을 처리하지 않고 넘어간다. 이렇게 넘어간 InProcess의 사용자 요청은 이전 리더가 장애 상황을 극복하고 팔로워로 재시작할 때, 자신이 이전 리더였고 사용자 요청을 처리하는 중에 중지되었음을 확인하게 해주는 정보가 된다.

[line 13-15 in table.2] 레지스터 서버는 ETCD에서 가져온 한 사용자 요청에 대한 처리가 완료되면, 자신이 마지막으로 처리한 사용자 요청에 대한 인덱스를 나타내는 LPI를 해당 요청에 대한 인덱스로 업데이트해 준다. 마지막으로 자신의 LPI와 ETCD에 저장된 사용자 요청의 총 개수를 나타내는 LS의 값이 같아지면, BR2K 최신 서비스의 상태에 대한 인덱스를 나타내는 SSI를 LS의 값으로 업데이트하여 최신 서비스 상태를 변경하고 LS 업데이트 이벤트에 대한 리스너 스트림을 생성하여 해당 액션을 종료한다.

prepare service 액션을 수행한 레지스터 서버는 서비스를 제공하는 리더 역할을 수행할 준비가 완료되었기 때문에, 이더리움 블록체인 기반으로 실행되는 서비스 레지스트리에 자신의 접속 정보를 최신 서비스 접속 정보로 업데이트한다. 이 작업으로 수 만개의 노드로 구성된 이더리움 블록체인에 최신 서비스 접속 정보가 각 노드에 복제되고 이로 인하여 서비스 사용자는 이더리움 블록체인을 구성하는 임의의 노드에 접근하더라도 새로운 서비스 접속 정보를 가져올 수 있어, BR2K 서비스는 신속하게 서비스를 재개할 수 있다.

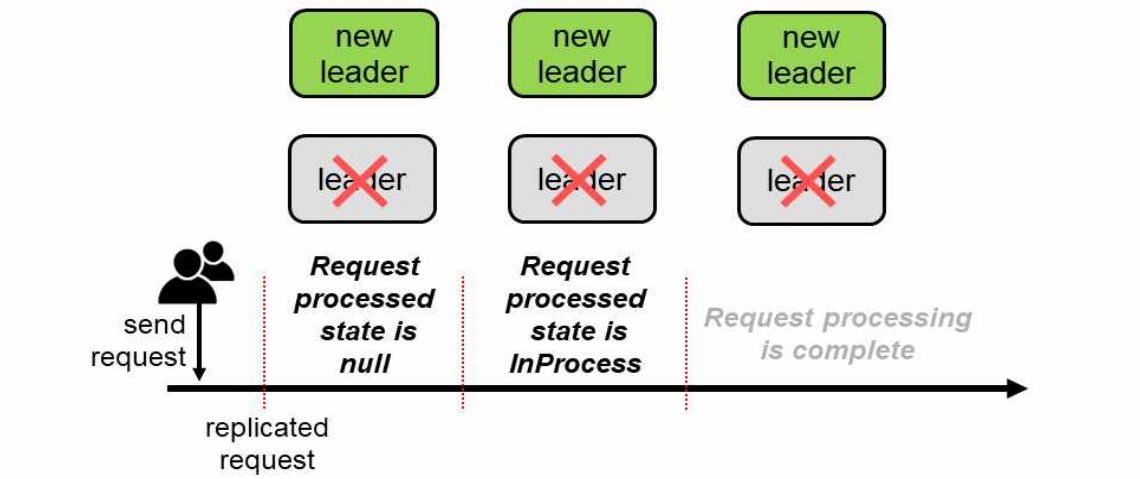

그림 8. 리더가 중지되는 3가지 시점

BR2K 서비스가 장애 상황에서도 일관성을 보장하는지를 확인하기 위하여, 이전 리더가 어떠한 시점에 중지되었는지를 파악해야 한다. 그림 8은 장애 상황에 의하여 중지된 리더의 3가지 시점을 나타낸다. BR2K 서비스에서 리더가 중지되는 시점은 크게 리더가 ETCD를 이용하여 사용자 요청을 팔로워들에게 복제만 한 경우(§fig.8, Request state is null), 복제된 사용자 요청을 처리하는 중에 중지되는 경우(§fig.8, Request state is InProcess), 사용자 요청에 대한 처리가 끝난 경우로(§fig.8, Request processing is complete) 나뉜다. 이와 같은 모든 시점에서 BR2K 서비스가 일관성을 보장할 수 없게 되는 시점은 이전 리더가 사용자 요청을 처리하는 중에 중지되는 경우이다. 이 경우에는 이전 리더가 로컬 상태에 대한 변경 작업을 수행하는 중에 중단되었기 때문에, 팔로워 서버의 로컬 상태와 일치하지 않을 수 있어 상태에 대한 일관성을 보장할 수 없게 된다.

장애 상황을 극복한 이전 리더는 재시작되면, 팔로워 역할로 전환된다. 서버가 팔로워 역할로 전환되면 *Completion of processing request* 액션이(§fig. 5) 시작된다. 이 액션은 서버가 이전 역할에서 리더였고 요청을 처리하는 중에 중지하였는지를 검사한다. 예를 들어, 그림 9와 같이 팔로워는 이전 리더 역할에서 인덱스가 43인 사용자 요청을 처리한 후 인덱스 44인 사용자 요청을 처리하는 과정에서 중지되었다면, 자신의 LPI의 값은 43 인덱스를 가진다. 그리고 Completion of processing request 액션에 의하여, 팔로워 서버는 다음에 처리해야 하는 44 인덱스인 사용자 요청에 대한 RS를 확인하여 *Inconsistent condition*을 검사한다. 이 conditon은 이 RS에 기록된 처리 상태가 InProcess이고 자신의 서버 ID가 있다면 True가 되고, 아니라면 False가 된다. 이때, Inconsistent condition이 True인 팔로워는 이전 리더 역할에서 자신이 사용자 요청을 처리하는 중에 중지되어 상태에 대한 일관성을 보장하지 못할 가능성이 있다고 스스로 판단하여 러너 역할로 전환된다.

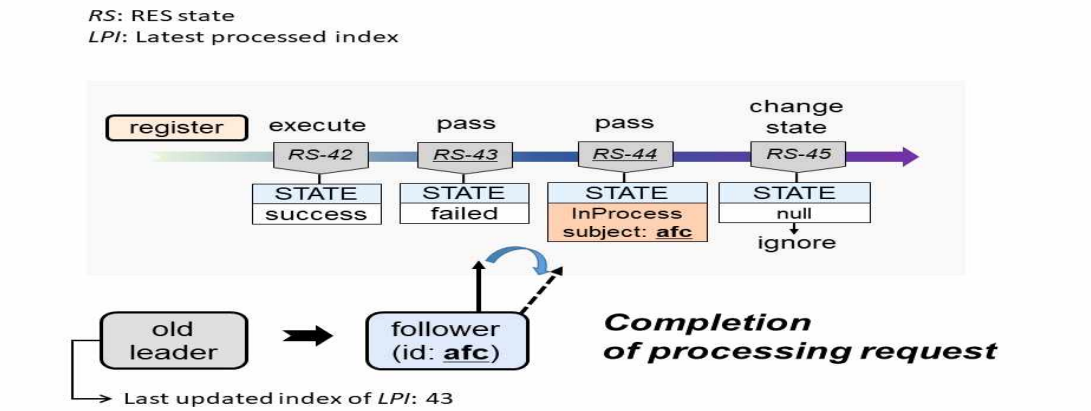

그림 9. Completion of processing request 액션

러너가 된 서버는 한 사용자 요청 처리에 대한 완결성을 가지지 못하였고 이로 인하여 이 서버의 로컬 상태가 다른 서버의 로컬 상태와 동기화되지 못할 수 있다는 걸 의미한다. 이렇게 판단되는 이유는 그림 10과 같이 한 사용자 요청에 대한 RS의 상태를 InProcess로 업데이트하고 중지되면 리더가 중지되는 시점이 3가지로 나누어지기 때문이다.

(1) 리더가 RS에 대한 상태만 InProcess로 업데이트했지만, 사용자 요청을 처리하지 못하고 중지될 때
(2) 리더가 사용자 요청에 대하여 처리하는 중에 중지될 때
(3) 리더가 사용자 요청에 대한 처리를 완료했지만, 처리에 대한 결과값을 RS의 상태에 반영하지 못하고 중지될 때

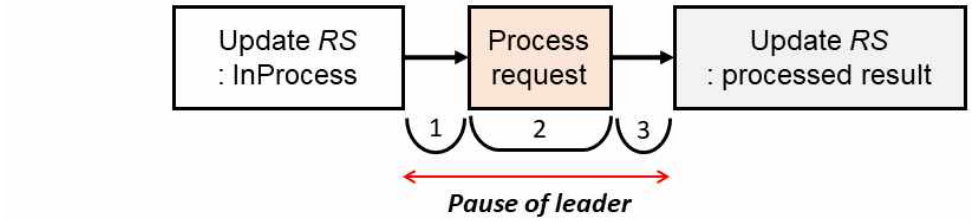

그림 10. 사용자 요청의 처리 과정에 대한 3가지 시점

InProcess 업데이트 후, 중지된 서버는 실제로 사용자 요청을 처리했는지에 대한 여부를 예측할 수 없으며 리더가 사용자 요청에 대하여 실제로 처리하는 중에 중지되었다면 어떠한 작업까지 수행하여 로컬 상태를 변경하였는지 확인할 수 없다. 이러한 이유로 해당 서버는 우선적으로 팔로워 역할에 대한 액션을 수행하지 않고 러너 역할로 전환되어 서비스 상태에 대한 복구 작업이 이루어져야한다.

3.2.2 상태에 대한 백업 및 복구

BR2K 서비스에는 한 서버의 로컬 상태에 대한 복구 작업, BR2K 서비스 전체에 대한 복구 작업 그리고 서비스가 지속함에 따라 ETCD에 쌓이는 사용자 요청으로 인한 ETCD 용량을 초기화해야 하는 작업 때문에, BR2K 서비스에서는 최신 서비스 상태에 대한 백업 작업이 필요하다. 이 백업 작업은 리더 서버의 *State version up* 액션에 (§fig. 5) 의하여 수행된다. BR2K 서비스에서는 ETCD의 용량을 초기화하고 서비스의 상태에 대한 백업이 몇 번 수행하였는지를 나타내는 상태 버전이 존재한다.

이 상태에 대한 버전은 BR2K 서비스의 전체 상태 버전을 나타내는 *SV* (§table. 1, State version)와 각 서버의 로컬 상태 버전을 나타내는 *SSV* (§table. 1, Server state version)로 구분되며 ETCD에 저장하여 관리한다. BR2K 서비스에서 서비스가 처음 배포될 때 모든 상태의 버전은 1로 시작되며 백업 작업을 수행하기 위한 *State version up* 액션이(§fig. 5) 수행될 때마다 1씩 증가한다.

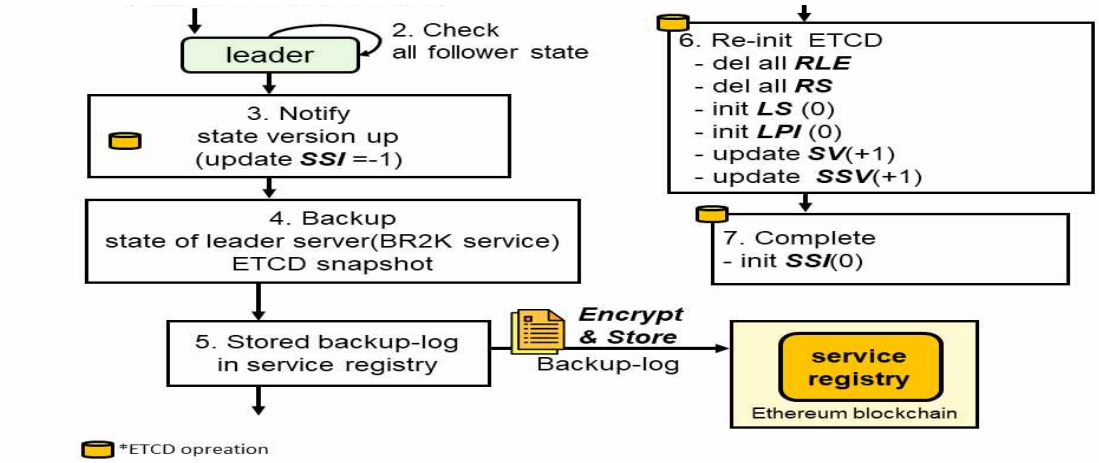

그림 11. 백업 및 용량 정리 과정

State version up 액션은 3가지 경우에 의해서 시작된다. 첫 번째는 BR2K 서비스의 배포자가 ETCD의 용량을 고려하여 지정한 최대 사용자 요청 개수만큼 사용자 요청이 복제될 때이다. 두 번째는 특정 주기가 될 때마다. 마지막은 중지된 서버가 과반수 이상일 때 시작된다. 이 액션을 시작한 리더는 사용자에게 요청을 더 이상 받지 않고, 현재 ETCD에 의하여 복제된 사용자 요청까지만 처리한다. 그리고 리더는 각 팔로워의 LPI와 BR2K 서비스의 상태를 나타내는 SSI가 같은지를 비교하여, 그림 11에 2번 과정처럼 모든 팔로워가 최신 서비스 상태인지를 지속적으로 확인한다. 모든 팔로워가 최신 서비스 상태를 가진다면, 리더는 SSI를 -1로 업데이트하여 모든 팔로워에게 백업 작업과 ETCD의 용량 작업을 시작한다고 알린다.(§fig. 11, Notify state version up) 각 팔로워는 최신 서비스 상태를 따라가는 the latest service state 액션 과정에서 SSI가 -1로 변경됨을 확인하여 해당 액션을 중지하고, 리더의 state version up 액션이 종료될 때까지 기다린다.

표 3. 상태 백업 로그의 구조

종류	설명
start-time	시작된 시간
end-time	끝난 시간
version	BR2K 서비스 상태 버전
members	BR2K 서비스를 구성하는 각 서버의 ID 리스트
etcd snapshot size	ETCD의 스냅샷 크기
service state size	서비스 상태 크기
storage location	오프체인 스토리지의 위치
storage access key	오프체인 스토리지의 접근키
subject	백업 작업을 수행한 리더 ID

State version up 액션에 대한 백업 작업은 현재 리더 서버의 실행 노드에 있는 로컬 상태에 대한 파일들과 데이터 복구 작업에 쓰이는 데이터인 ETCD의 스냅샷(Snapshot)을 오프체인 스토리지(Off-chain storage) 기록한다. (§fig. 11, Backup state of leader service, ETCD snapshot) 오프체인은 블록체인 이외의 외부 공간에 대용량 데이터를 특정 스토리지에 기록하는 방식이며, 본 논문의 기법에서는 공개형 블록체인에(Pulbic blockchain) 대용량 파일을 저장하는 것은 성능이 낮고 합의에 의한 큰 비용이 들기 때문에 오프체인 스토리지 방식을 사용한다.

리더는 백업 작업이 완료되며 상태 및 서비스 복구 작업을 수행할 때 이용되는 백업 작업에 대한 로그를(§table.3) 생성하고 암호화하여 이더리움 블록체인에 실행되어 중요한 정보를 안전하게 관리할 수 있는 서비스 레지스트리에 저장한다. 이때, 백업 로그를 BR2K 서비스가 소유한 이더리움 블록체인 계좌의 공개 키(Public key)로 암호화하고 비밀 키로만(Private key) 해당 로그를 복호화 할 수 있게 하여 권한이 없는 주체가 열지 못하도록 한다.

서비스 레지스트리에 백업 로그를 저장하는 작업이 완료되면 리더는 현재 ETCD에 저장된 사용자 요청인 RLE와 사용자 요청의 처리 상태 값인 RS를 모두 삭제하여 용량을 확보하고 사용자 요청에 대한 로그 사이즈인 LS와 리더의 LPI를 0으로 초기화한다. 또한, BR2K 서비스의 상태 버전과(SV) 리더의 상태 버전을(SSV of leader) 1 증가시키는 작업을 수행한다. 모든 작업이 완료되면 리더는 각 팔로워에게 state version up 액션이 종료되었다는 걸 SSI를 0으로 업데이트하여 알린다. 대기 상태였던 팔로워는 SSI의 업데이트 이벤트를 통해 백업 작업이 종료되었음을 확인하여 자신의 LPI를 0으로 초기화하고 자신의 상태 버전을(SSV of follower) 1 증가시키는 작업을 수행하며 the latest service state 액션을 다시 시작한다.

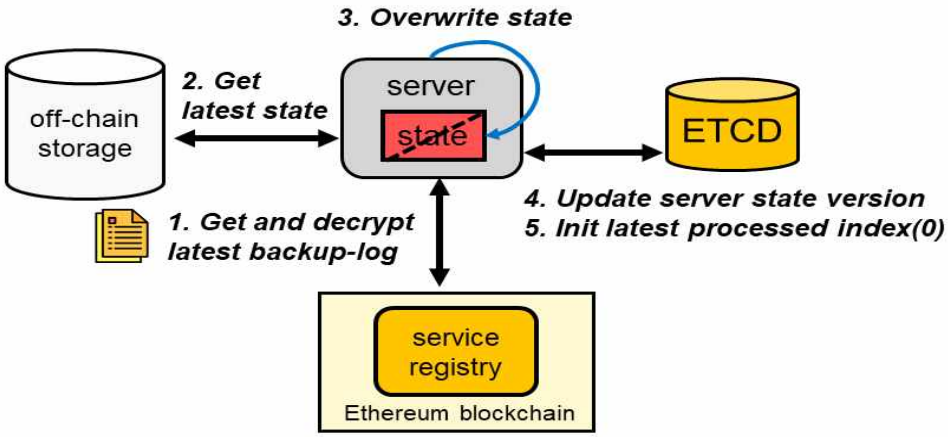

그림 12. 서비스 상태 복구 과정

리더의 State version up 액션에 의하여 상태에 대한 백업이 완료되면, 러너 서버는 일관성을 보장하지 못한다고 여겨지는 상태에 대한 복구 작업을 수행할 수 있다. 이 작업은 *Recovery state* 액션이라고(§fig. 5) 하며, 상태에 대한 복구 작업뿐만 아니라 BR2K 서비스에 새로운 서버가 합류했을 때 최신 서비스의 상태를 빠르게 따라가기 위한 방법으로 사용될 수 있다. 그림 12는 서버의 로컬 상태에 대한 복구 과정을 나타낸다. 복구 작업에 대한 절차는 다음과 같다.

① 서비스 레지스트리에 암호화된 백업 로그를 가져와 복호화 한다.
② 복호화된 백업 로그를 이용하여 오프체인 스토리지에 접근하고 백업된 최신 상태를 가져온다.
③ 서버의 현재 로컬 상태를 오프체인 스토리지에서 가져온 로컬 상태로 변경한다.
④ 서버의 상태 버전을 현재 서비스의 상태 버전으로 업데이트한다
⑤ 서버의 LPI를 0으로 초기화한다.

4. BR2K 기법의 신속하고 체계적인 배포 및 복구

BR2K 서비스는 BR2K 기법이 적용된 블록체인 응용 서비스이며, 서비스를 제공하는 하나의 리더와 리더가 서비스를 제공할 수 없을 시에 대비한 다수의 팔로워로 구성되어 복제된다. 본 절에서 BR2K 서비스를 일관되고 신속하게 분산 환경에 배포하기 위하여, 쿠버네티스 기반의 배포 기법에 대하여 설명한다. 또한, BR2K 서비스가 과반수의 서버가 중지되어 서비스 자체가 중지될 때 신속하고 체계적인 복구를 지원하는 방법에 대하여 자세히 기술한다.

4.1 BR2K 기법의 서비스 배포

BR2K 서비스는 블록체인 응용 서비스를 복제하기 위하여 분산 환경에 서버를 중복하여 실행한다. 이때, 분산 환경을 구성하는 모든 노드가 동일한 환경을 가지지 못한다면, 서비스 배포 및 운영하는 과정에서 예측할 수 없는 문제가 발생할 수 있다. 또한 실제 개발 환경에서분산 환경을 구성하고 블록체인 응용 서비스의 서버를 중복하여 실행하는 작업은 번거롭고 쉽지 않으며 이는 복구 작업을 수행할 때 걸리는 시간에 많은 영향을 미친다. 본 절에서는 BR2K 서비스를 분산 환경에서 일관성 있는 형태로 신속하게 배포하기 위하여, 분산 컨테이너 관리 도구인 쿠버네티스 기반을 이용한 BR2K 기법의 배포에 대하여 설명한다.

4.1.1 쿠버네티스 기반의 신속한 배포

BR2K 서비스를 모든 환경에서 멱등성 있는 형태로 실행하고 분산 환경에 신속하게 배포하기 위하여, BR2K 기법에서는 블록체인 응용 서비스와 ETCD를 도커와 같은 컨테이너 플랫폼을 이용하여 컨테이너 이미지의 형태로 빌드하고 이를 분산 컨테이너 관리 도구인 쿠버네티스를 이용하여 배포한다. 그러나 일반적인 쿠버네티스 환경에서는 BR2K 서비스의 구조에 맞게 쿠버네티스에 즉각적으로 배포할 수 없다.

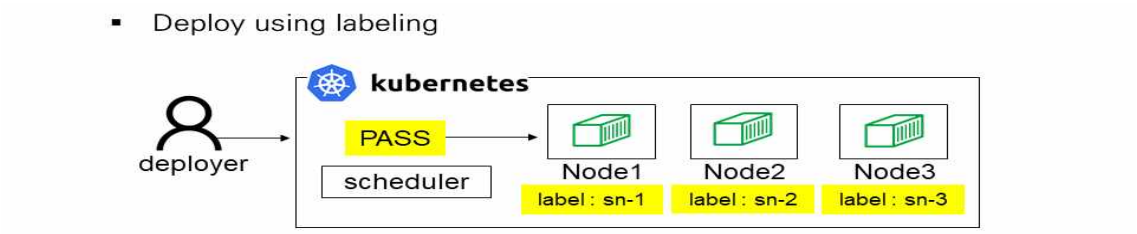

그림 13. 스케줄러 기반의 배포와 라벨링 기반의 배포

쿠버네티스에서 스케줄러를 이용한 일반적인 방식으로 BR2K 서비스를 배포하게 되면, 그림 13과 같이 하나의 노드에 두 개 이상의 서버가 실행되어 BR2K 서비스의 구성에 맞지 않은 상태로 배포된다. 그러므로 BR2K 서비스를 쿠버네티스에서 구성하려면 쿠버네티스를 구성하는 각 노드에 라벨링 작업이 필요하다. 라벨링 작업은 그림 13과 같이 키-값으로 구성된 고유한 라벨을 BR2K 서비스가 배포될 쿠버네티스의 각 워커 노드에 할당하는 작업이다. 이 작업을 완료한 쿠버네티스에서는 서비스를 배포할 때, 스케줄링 방식이 아닌 하나의 워커 노드를 직접 선택하여 하나의 서버가 실행되도록 할 수 있다. BR2K에서 쿠버네티스 기반의 배포 기법은 이러한 기능을 사용하여, 각 노드에 하나의 블록체인 응용 서비스의 서버와 ETCD가 실행되도록 한다.

표 4. 컨텍스트 구조

종류	설명
k8s-locations	쿠버네티스의 접근 위치 리스트
access-token	서비스를 배포하기 위한 액세스 토큰
node-id	배포되는 서비스가 쿠버네티스에서 사용하는 자원에 대한 ID
network-id	배포되는 서비스가 사용하는 네트워크의 ID
service-protocol	배포되는 서비스가 사용하는 네트워크 프로토콜 타입(HTTP, HTTPS, GRPC)
service-points	배포되는 서비스가 사용하는 엔드포인트 리스트
service-port	배포되는 서비스가 사용하는 포트 번호
node-mount-path (service)	배포되는 서비스의 상태에 대한 저장 공간
etcd-mount-path (etcd)	배포되는 서비스가 사용하는 ETCD에 대한 저장 공간

위와 같은 라벨링 작업 외에도, BR2K 서비스를 쿠버네티스에 배포하기 위해서는 쿠버네티스의 리소스들이 필요하다. 배포에 필요한 쿠버네티스의 리소스에는 서비스가 실행되는 논리적인 작업 공간인 네임스페이스(Namespace), 배포자에 대한 인증 및 권한에 대한 액세스 토큰(Access Token), 사용할 수 있는 워커 노드의 개수 그리고 서비스에 대한 엔드 포인트 작업을 수행하는 서비스 오브젝트(Service Object) 등이 존재한다. 이러한 BR2K 서비스를 배포하기 위한 쿠버네티스에서 필요한 리소스들이 명세된 파일을 BR2K 기법에서는 컨텍스트(Context)라고 명명한다. 표 4는 컨텍스트 구조를 나타낸다.

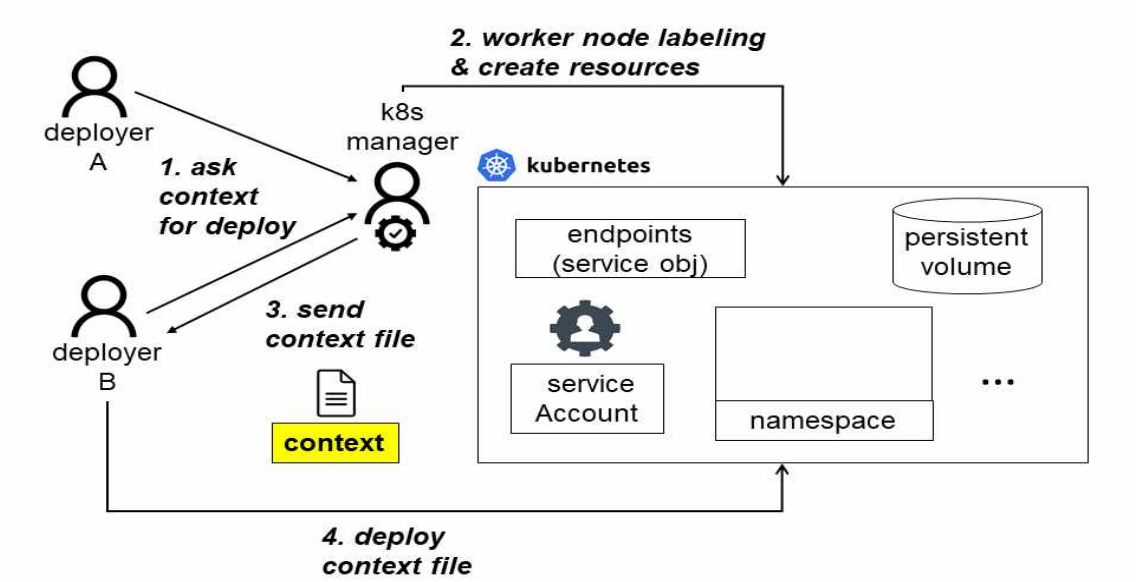

그림 14. 쿠버네티스를 이용한 BR2K 서비스 배포 과정

BR2K 기법에서는 쿠버네티스의 전반적인 관리를 수행하는 관리자와 BR2K 기법이 적용된 서비스를 배포하는 배포자들이 존재한다. 그림 14와 같이 BR2K 서비스에 대한 배포자들은 쿠버네티스 관리자에게 해당 BR2K 서비스 배포에 필요한 최대 노드의 개수, 외부 액세스를 위한 포트 및 도메인 네임, 서비스 이름과 같은 정보를 포함하여 쿠버네티스 관리자에게 컨텍스트 파일을 요청을 한다. 쿠버네티스 관리자는 배포자가 보내온 정보를 기반으로 해당 서비스에 적합한 워커노드에 대한 라벨링 작업을 수행하고 각종 자원을 쿠버네티스에서 생성한다. 그리고 생성된 자원들과 라벨링된 워커 노드들에 대한 정보들을 나열하여 컨텍스트 파일을 만들어 배포자에게 보낸다. 컨텍스트를 발급 받은 배포자는 해당 컨텍스트만 있으면 언제든지 쿠버네티스 환경에 BR2K 서비스를 신속하고 일관된 배포를 할 수 있다.

4.2 BR2K 기법의 서비스 복구

지진, 홍수 및 테러와 같은 여러 가지 재난이나 다양한 해킹 공격으로 인하여 다수의 노드가 파괴되는 최악의 상황에서, 견고성을 지원하기 위하여 쿠버네티스 환경에서 복제 실행된 블록체인 응용 서비스는 체계적이고 신속하게 재가동할 수 있어야 한다. 본 절에서는 BR2K 서비스의 복구를 위하여 컨테이너 이미지를 안전하게 관리하기 위한 오픈소스 컨테이너 레지스트리 하버의 구성과 장애 복구 센터 역할을 수행하는 서비스 레지스트리를 이용한 체계적인 복구를 지원하는 방법에 대해 설명한다.

4.2.1 컨테이너 레지스트리의 구성

BR2K 기법에서 쿠버네티스에 BR2K 서비스를 배포하기 위해서는 쿠버네티스의 자원에 대한 명세 파일인 컨텍스트와 해당 서비스에 대한 컨테이너 파일인 이미지가 필요하다. 특히, 쿠버네티스 환경에서 실행되는 컨테이너 이미지는 분산 환경에서 복제되어 안전하고 신뢰할 수 있는 독립적인 저장소에서 관리되어야 한다. 그러므로 BR2K 기법은 BR2K 서비스가 빌드된 컨테이너 이미지의 저장을 대중적으로 많이 사용하는 개인형 컨테이너 레지스트리(Private container registry)인 하버(Harbor)를 사용한다.

하버는 일반적으로 컨테이너 이미지를 저장하고 관리하는 용도로 많이 사용하는 오픈 소스 컨테이너 레지스트리이며, 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.

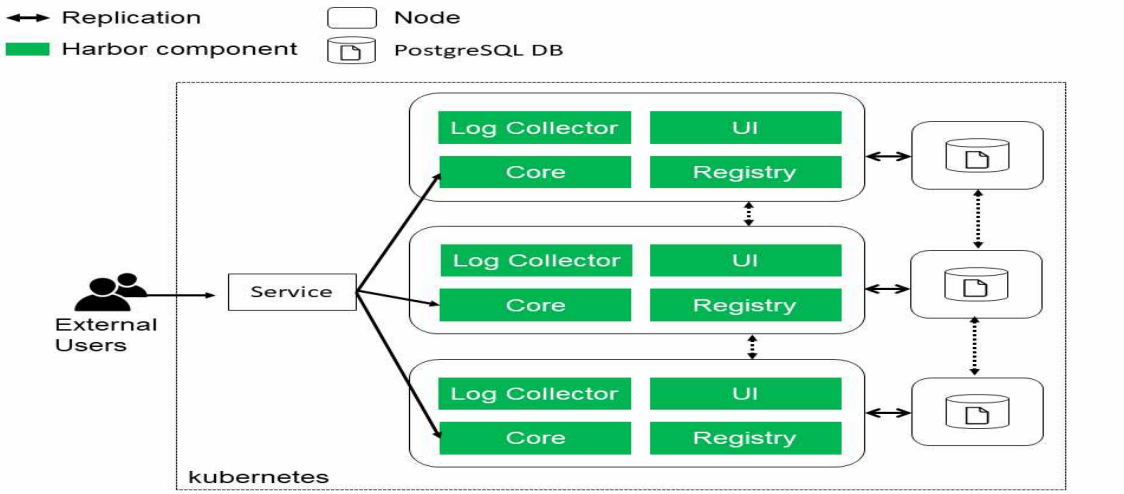

그림 15. 쿠버네티스 기반으로 복제된 하버 구조

BR2K 기법에서는 BR2K 서비스의 컨테이너 이미지를 보관하는 하버의 가용성을 높이기 위하여, 쿠버네티스 기반으로 하버를 복제하여 구축한다. 그림 15는 쿠버네티스에 배포된 컨테이너 레지스트리 하버의 구조이다. 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 맵핑 시켜, 외부 사용자가 하버를 이용할 수 있도록 한다.

4.2.2 서비스 레지스트리

서비스 레지스트리는 BR2K 기법의 서비스 복제가 적용된 블록체인의 응용 서비스를 지진, 테러와 같은 재난이나 다양한 해킹 같은 상황에서도 사용자에게 서비스의 접속 정보를 언제든지 제공하며 서비스 복구 정보를 관리하기 위한 이더리움 블록체인 기반의 스마트 컨트랙트이다. BR2K 서비스는 네트워크 파티션과 같은 장애 상황이 발생하면 서비스를 제공하는 새로운 리더를 선출하고 이 서비스 레지스트리에 새로운 서비스 접속 정보를 등록하는 작업을 수행하여, 사용자가 10,000개가 넘는 노드로 구성된 이더리움 공개 블록체인 환경에서 현재 서비스를 제공하는 위치를 언제든지 가져올 수 있도록 한다.

이와 같은 방식은 서비스를 제공하는 서버의 실패와 같은 상황에서도 분산 실행된 이더리움 블록체인 노드에서 BR2K 서비스 접근 위치를 언제든지 가져와, 보다 안정적으로 BR2K 서비스에 접근할 수 있는 장점을 가진다. 더불어, 그림 16과 같이 서비스 레지스트리에 서비스 관리자는 서비스 결함에 대한 회복을 위하여 서비스 메타 정보 및 복구 정보를 저장하여, 배포된 BR2K 서비스에 결함이 생기면 서비스 레지스트리에서 즉각적으로 복구 정보를 가져와 결함이 있는 서버를 복구할 수 있다.

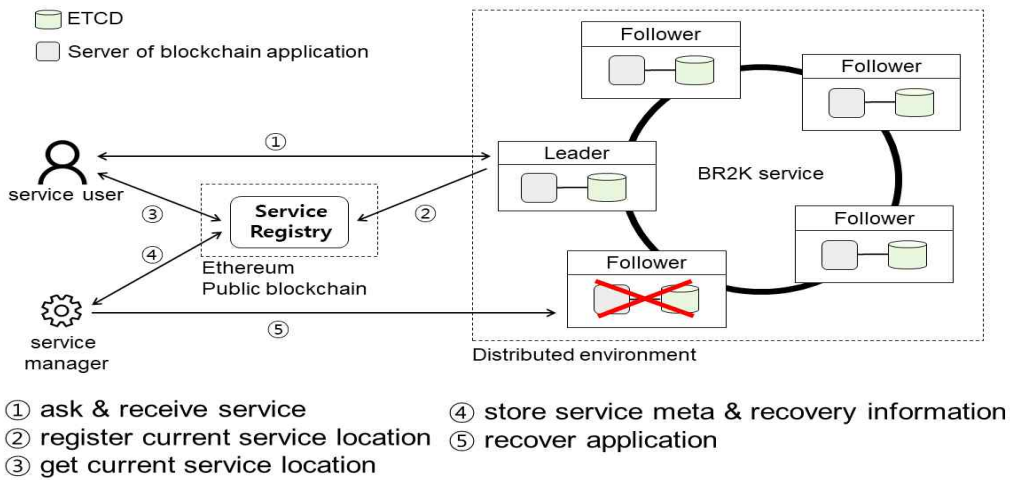

그림 16. BR2K 기법의 서비스 레지스트리

BR2K 기법의 서비스 레지스트리는 그림 17과 같은 구조를 가진다. 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, 블록체인 응용 서비스의 복제 수, 서비스 상태에 대한 백업 로그 그리고 BR2K 서비스의 컨테이너 이미지를 관리하는 하버의 위치 및 접근 계정과 같은 서비스 복구 정보가 저장된다. 이와 더불어 서비스 복구 요청 기능(§fig. 17, send Claim), 현재 서비스 위치 얻기(§fig. 17, getServiceLocation) 그리고 서비스 위치 업데이트와 복구 요청을 알리는 이벤트 기능(§fig. 17, NewLocation & Claim)과 같은 기존의 서비스 레지스트리의 모든 기능을 제공한다.

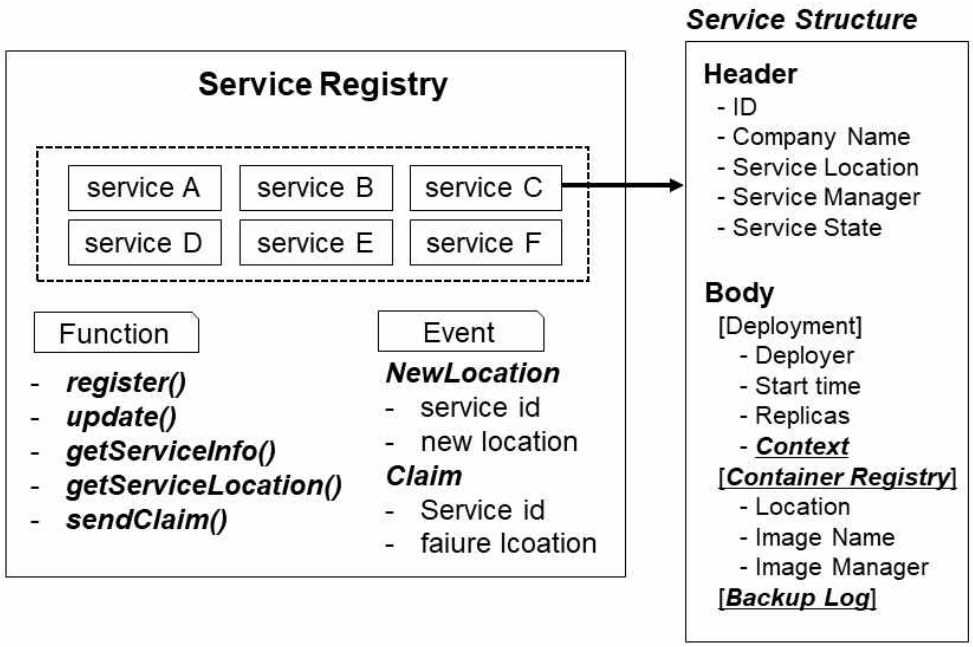

그림 17. 서비스 레지스트리의 구조

그림 18은 블록체인 서비스 레지스트리를 이용한 복구 절차를 나타낸다. BR2K 서비스는 그림 18과 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 sendClaim을 사용하여 관리자에게 복구 요청을 보낸다.

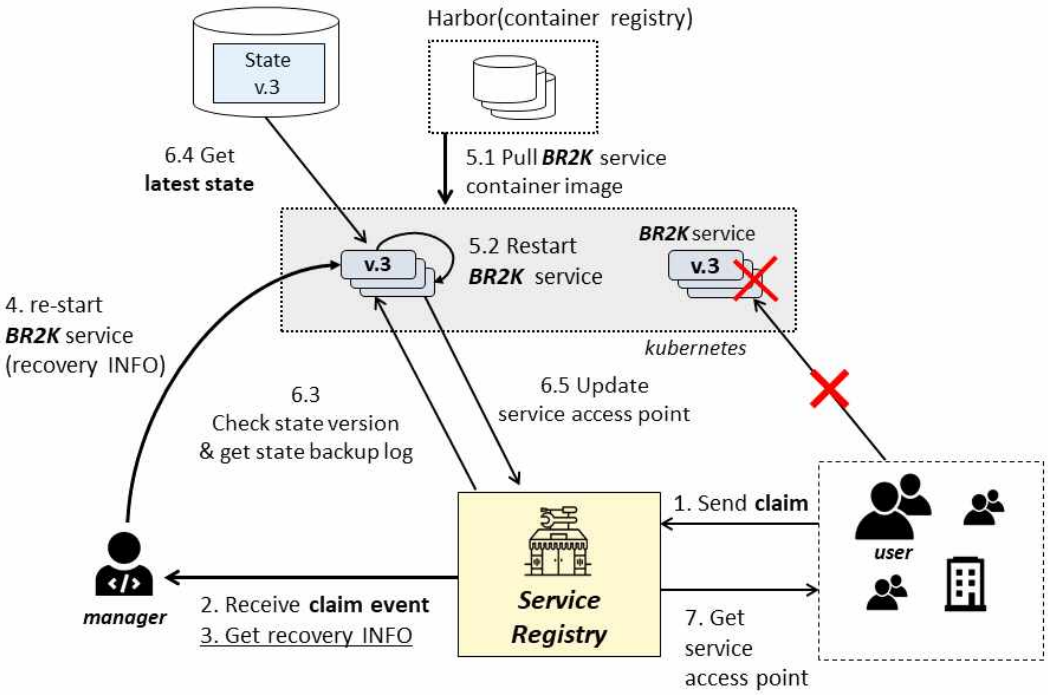

그림 18. 서비스 레지스트리를 이용한 BR2K 서비스 복구

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(컨텍스트, 서비스 상태에 대한 백업 로그, 컨테이너 이미지가 저장된 레지스트리 위치와 접근정보 등) 가져오고 이를 이용해 BR2K 서비스를 쿠버네티스에 재배포한다. 이때, BR2K 서비스가 재배포되는 과정에서 서비스 복구 정보에 있는 컨테이너 레지스트리인 하버 접근 정보와 백업 로그를 이용하여, BR2K 서비스의 컨테이너 이미지를 가져오고 오프체인 스토리지에서 최신 서비스 상태를 가져와 BR2K 서비스를 재실행하게 된다.

5. BR2K 기법을 위한 활용 도구

제안된 BR2K 기법의 서비스 복제 방법을 적용하여 블록체인 응용 서비스를 개발하는 작업, 개발된 서비스를 컨테이너 이미지로 빌드하는 작업, 쿠버네티스 환경에 BR2K 서비스를 배포하고, 중지하고, 다시 재배포하는 작업 그리고 복제 수 조절 등과 같은 작업은 실제 개발환경에서 하는 것은 쉽지 않다. 본 절에서는 실제 개발 환경에서 BR2K 기법을 손쉽게 적용하기 위하여 개발된 BR2K 프레임 워크를 설명한다. 또한 쿠버네티스에 배포된 BR2K 서비스들을 웹 브라우저 환경에서 지속적인 상태 확인, 복제 서비스의 결함에 대한 편리한 복구 등의 기능을 제공하는 모니터링 도구에 관해서 설명한다.

5.1 BR2K 서비스 개발 및 배포를 위한 프레임워크

BR2K 프레임워크는 BR2K 서비스에 대한 개발 및 컨테이너 이미지 빌드, 쿠버네티스 환경에서 서비스를 배포하기 위한 자원 명세 파일인 컨텍스트 요청 기능 그리고 개발된 BR2K 서비스를 쿠버네티스에 배포하는 기능 등을 제공하는 커맨드라인 인터페이스 기반의 프레임워크이다. 그림 19는 BR2K의 프레임워크 구조이다. 이 프레임워크는 대중적으로 사용되는 이더리움 블록체인 응용 서비스 개발 프레임워크인 트러플(Truffle)[20], 컨테이너 플랫폼인 도커(Docker) 그리고 서버 개발 프레임워크인 익스프레스(Express)를[21] 기반으로 확장하여 개발한다.

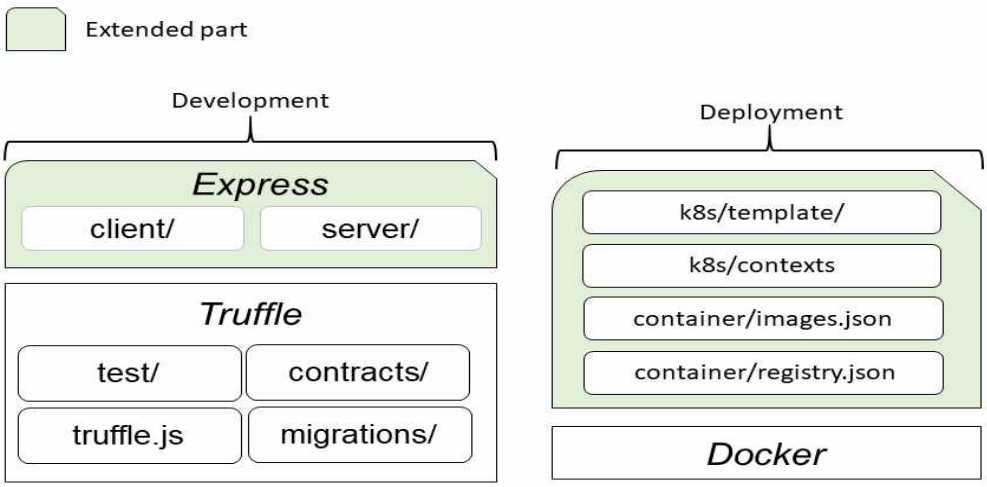

그림 19. BR2K 프레임워크 구조

BR2K 프레임워크 구조는 BR2K 서비스에 대한 개발과 배포로 나뉜다. 그림 19의 개발(Development) 파트는 익스프레스 프레임워크와 트러플 프레임워크 기반으로 이더리움의 스마트 컨트랙트 배포 및 테스트, BR2K 서비스 개발의 복잡성을 (§fig.19, client/ & server/) 추상화시켜준다. 이 프레임워크의 배포(Deployment) 파트에서는 BR2K 서비스에 대한 컨테이너 이미지 빌드 및 관리(§fig. 19, image.json), 빌드된 컨테이너 이미지를 컨테이너 레지스트리 하버에 저장 및 관리(§fig. 19, registry.json), 쿠버네티스에 BR2K 서비스 배포(§fig. 19, contexts)와 같은 작업을 수행할 수 있다. 그림 20은 BR2K의 프레임워크 커맨드라인에 대한 종류를 나타낸다.

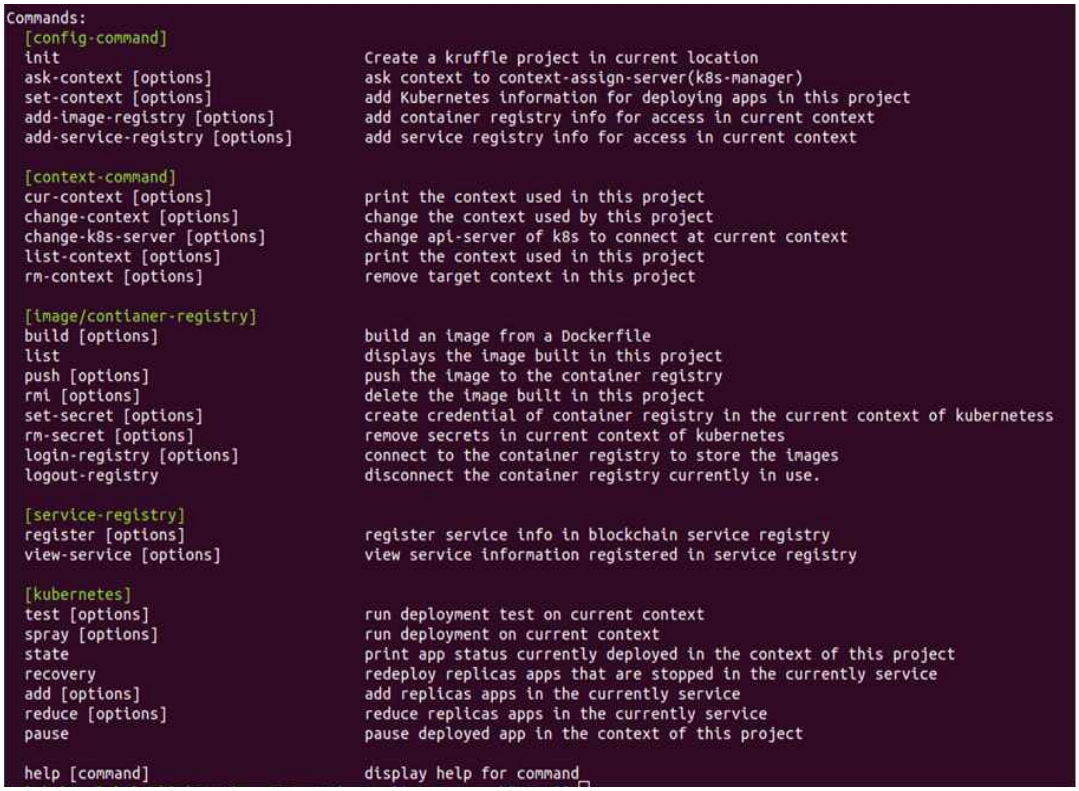

그림 20. BR2K 프레임워크 기능

그림 20과 같이 커맨드 라인 인터페이스 기반의 BR2K 프레임워크를 이용하여 실제 개발 환경에서 BR2K 서비스가 배포되는 과정은 다음과 같다.

① BR2K 프레임워크 프로젝트 생성을 한다. (§fig. 20, init 커맨드) 그러면 그림 19에 있는 프로젝트 파일이 생성된다.

② 쿠버네티스 관리자한테 해당 BR2K 서비스에 대한 컨텍스트를 요청한다. (§fig. 20, ask-context 커맨드)

③ 생성된 프로젝트에 포함된 BR2K 서비스 개발에 대한 프레임워크 모듈을 이용하여 서비스를 개발한다.

④ 생성된 프로젝트에 발급받은 컨텍스트 파일, 서비스 레지스트리와 컨테이너 레지스트리 하버에 접근하기 위한 계정 정보를 설정한다. (§fig. 20, set-context, add-image-resigtry, add-service-registry 커맨드)

⑤ 개발된 BR2K 서비스를 컨테이너 이미지로 빌드하고 컨테이너 레지스트리인 하버에 저장한다.(§fig. 20, build, push 커맨드)

⑥ 개발된 BR2K 서비스에 대한 백업 정보인 서비스 메타 정보, 저장된 서비스의 컨테이너 이미지에 대한 정보들을 서비스 레지스트리에 저장한다.(§fig. 20, register 커맨드)

컨테이너 이미지로 빌드하고 컨테이너 레지스트리인 하버에 저장한다.(§fig. 20, build, push 커맨드)

⑦ 개발된 BR2K 서비스를 배포한다. (§fig. 20, spary 커맨드)

BR2K 프레임워크에 포함된 BR2K 서비스에 대한 개발 프레임워크는 node.js 환경에서 대중적으로 사용하는 서버 개발 프레임워크인 익스프레스(Express) 기반으로 개발되어, 그림 21과 같이 기존의 익스프레스 프레임워크 방식으로 손쉽게 BR2K 기법을 적용할 수 있다.

```
const app = require('br2k-app')(RUNTIME_CONFIG)

app.replicate('POST', '/user', (request, response)=>{

})

app.onlyOnce('GET', '/user', (request, response)=>{

})
```

그림 21. BR2K 서비스 개발 프레임 워크

예를 들어, BR2K 서비스에서 replication 타입의 사용자 요청을 처리하는 부분의

개발은 그림 21에 개발 프레임워크 모듈인 app에 있는 replicate 함수를 이용하면 된다. 이 함수에 첫 번째 매개변수에 http(s)의 메서드 타입, 두 번째 매개변수에 해당 요청을 처리할 restful api 경로, 세 번째 매개변수에는 실제 사용자 요청을 처리하고 사용자에게 응답해주는 작업을 해주는 콜백 함수를 작성하여 BR2K 서비스를 개발한다.

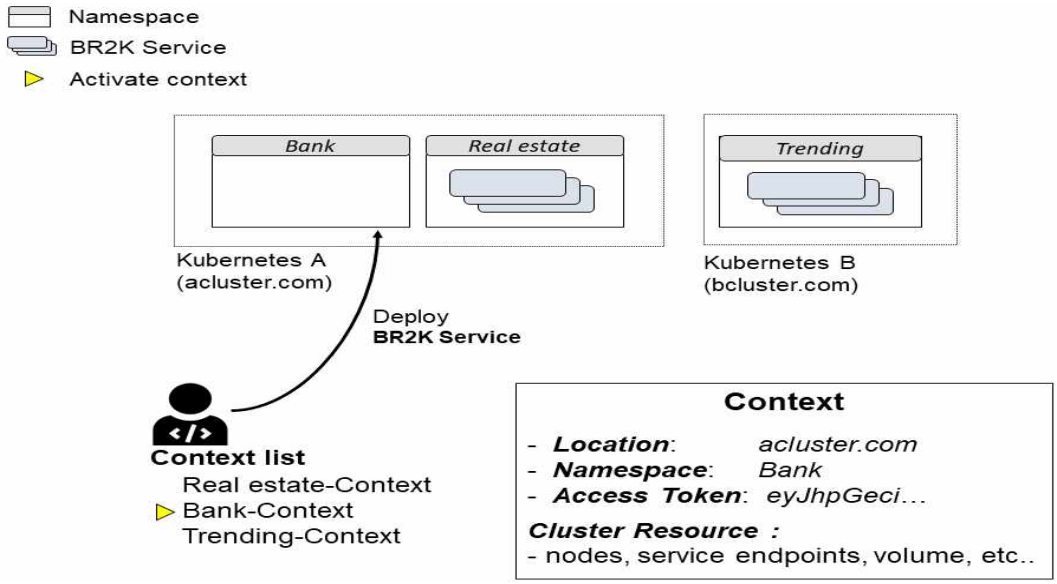

그림 22. BR2K 프레임워크의 다중 컨텍스트

또한, BR2K 프레임워크를 사용하는 서비스 배포자는 쿠버네티스의 관리자로부터 한 개 이상의 컨텍스트 정보를 할당받고 그림 22와 같이 생성된 BR2K 프로젝트에 컨텍스트 리스트에 저장할 수 있다. 그리고 서비스 배포자는 컨텍스트 리스트에 있는 컨텍스트들 중 하나를 선택하여 활성화하고 프레임워크에서 구현된 배포 커맨드를 이용하여 쿠버네티스에 BR2K 서비스를 신속히 배포할 수 있다.

5.2 BR2K 서비스 모니터링 도구

본 논문에서 BR2K 기법을 이용하여 복제한 블록체인 응용 서비스의 지속적인 상태 확인, 복제 서비스의 편리한 결함 복구 등의 기능을 제공하는 복제 상태 모니터링 도구는 블록체인 웹 응용 서비스인 BR2K Watch라고 명명한다.

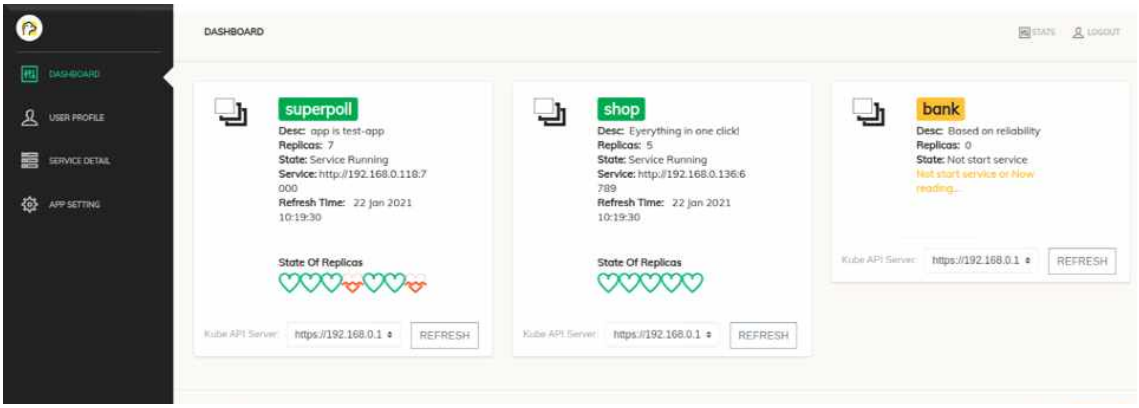

그림 23. BR2K Watch의 메인 화면

BR2K Watch는 웹 서버 프레임 워크인 Vuejs[22] 기반으로 구현되며 BR2K 기법의 서비스 레지스트리 기반으로 동작한다. BR2K Watch의 기능은 그림 23과 같이 서비스 레지스트리에 등록된 서비스 메타 정보(서비스의 이름, 설명, 실행 상태, 복제 개수 그리고 서비스 접근 위치)의 출력, 쿠버네티스에서 실행 중인 BR2K 서비스 상태 확인, 각 복제 서비스의 로그 출력, 복제 수 조절, 그리고 간편한 결함 복구가 있다. 서비스 관리자인 BR2K Watch의 사용자는 BR2K Watch를 이용하기 위하여, 서비스 레지스트리에 서비스 정보를 등록할 때 사용한 이더리움 계정으로 로그인해야 한다. BR2K Watch의 로그인 UI 화면에 사용자가 이더리움 계정과 해당 계정의 비밀키를 입력하여 로그인 버튼을 클릭하면, 그림 24와 같은 과정을 거쳐 사용자가 배포한 서비스들의 현재 상태를 BR2K Watch의 대시보드에 출력한다.

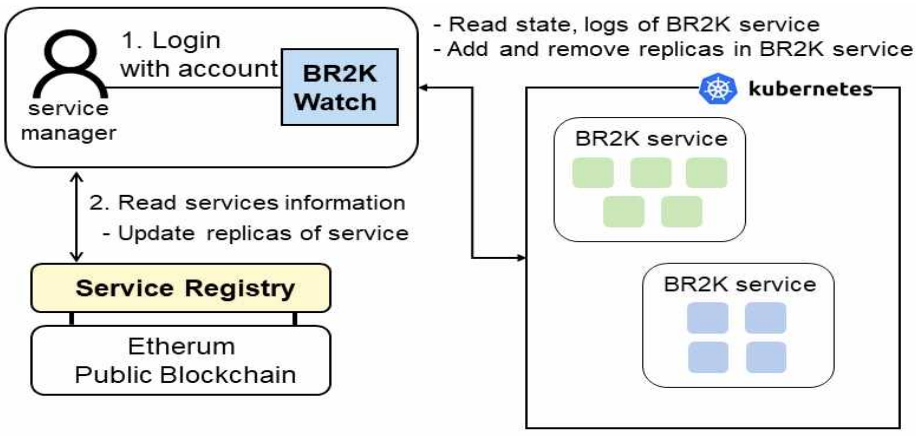

그림 24. BR2K Watch의 동작 과정

로그인 과정 후에 대시보드에 출력된 서비스의 이름을 사용자가 클릭하게 되면, 그림 25와 같이 BR2K Watch의 앱 세팅(App setting) 화면으로 이동할 수 있다. BR2K Watch의 앱 세팅 화면에서는 해당 서비스에 대한 각 복제 응용 서비스의 로그 출력, 서비스의 복제 응용서비스의 개수 조절 및 결함 복구 기능을 사용할 수 있다. 로그 출력은 사용자가 서비스에 현재 실행되고 있는 각 복제 응용 서비스를 선택할 때마다, 쿠버네티스에서 선택된 복제 응용 서비스의 실시간 로그를 출력할 수 있는 로그 스트림(Log stream)을 가져와 대시보드에 로그를 출력하는 기능이다.

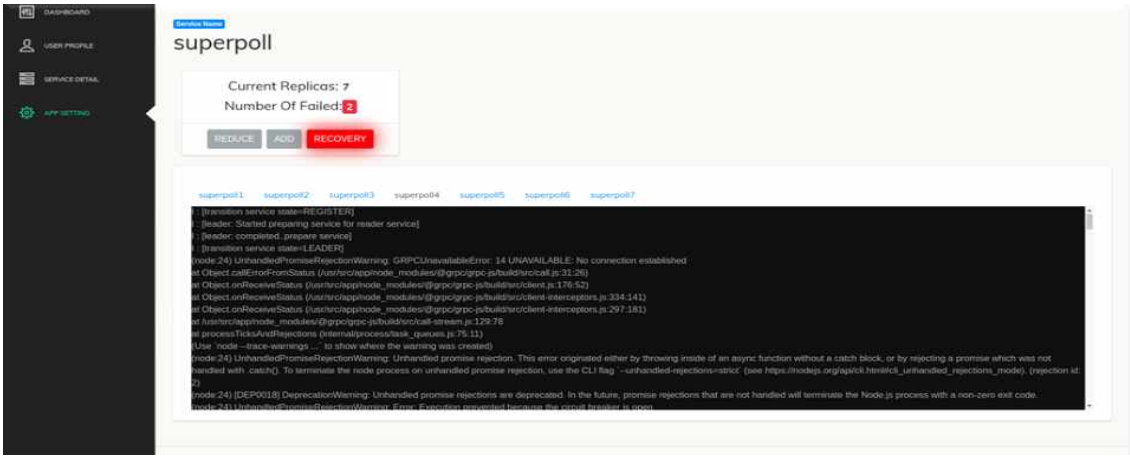

그림 25. BR2K Watch의 셋팅 메뉴

서비스의 복제 응용 서비스의 개수 조절 기능은 해당 서비스의 서비스 레지스트리에 등록된 최대 복제 수 범위만큼 복제 수를 조절할 수 있는 기능이다. 해당 기능은 복제 응용 서비스의 개수 조절 버튼을(fig. 25, REDUCE, ADD button) 통해 실행되며, 서비스 레지스트리에서 가져온 서비스 정보를 이용하여 쿠버네티스에 해당 서비스의 응용 서비스의 개수를 늘리거나 줄이고 서비스 레지스트리에 해당 서비스의 복제 응용 서비스 개수를 업데이트하는 과정을 거친다. 만약, 서비스의 현재 복제 상태에 문제가 있다면 복제 응용 서비스의 개수 조절 버튼이 비활성화되어 개수 조절 기능을 사용할 수 없게 되며 결함이 있는 서비스의 복제 응용 서비스에 대한 복구 작업이 필요하다. BR2K Watch의 결함 복구 기능은 서비스의 현재 복제 상태에 문제가 있을 때만 사용 가능하며 복구 버튼(fig. 25, RECOVERY button)을 통해 실행된다.

6. 적용 및 테스트

본 장에서는 블록체인을 이용하여 사용자의 정보를 신뢰성 있게 관리하기 위하여 개발된 DID(Decentralized Identifiers)[23] 기반의 사용자 정보 관리 서비스인 InfoDID를[24] 소개하고 이 서비스에 BR2K 기법을 적용하여 해당 기법의 서비스 복제 실행의 유효성을 검증하기 위한 테스트를 기술한다.

6.1 적용

최근에는 무결성, 탈중앙화, 높은 보안성 등의 특징을 가진 블록체인 기술을 이용하여 중앙화된 신원 제공자, 인증기관 등으로부터 벗어난 검증 가능한 분산 식별자 DID가 차세대 신원 인증 기술로 등장하였고 W3C(World Wide Web Consortium)에서 표준화를 시도하고 있으며[25] 이를 이용한 여러 가지 응용 서비스들이 개발되고 있다. 이러한 DID를 이용한 개발된 블록체인 응용 서비스들 중 하나인 InfoDID는 블록체인 기반의 DID 기술을 활용하여 빈번히 사용되는 개인 정보를 신뢰성 있게 관리하고, 사용자가 자신의 정보를 편리하게 다른 서비스에게 제공하도록 지원하는 서비스이다.

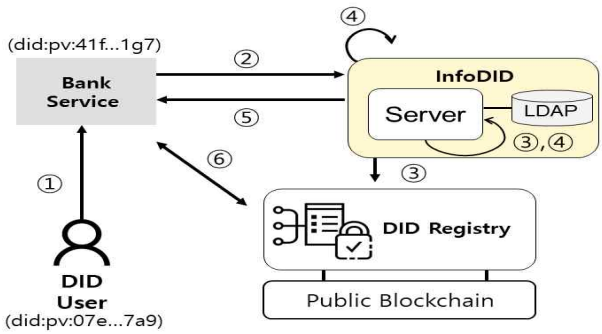

그림 26. InfoDID 서비스 시나리오

InfoDID는 블록체인 기반의 사용자 정보 저장의 비용 및 용량 한계 때문에 사용자 정보를 그림 26과 같이 LDAP(Lightweight Directory Access Protocol) 기반의 데이터베이스에 저장하여 관리한다. 또한 해당 서비스는 사용자에 대한 DID 인증을 하기 위한 이더리움 스마트 컨트랙트 기반의 DID 레지스트리 및 LDAP 기반의 데이터베이스와 상호작용하여 사용자에게 서비스를 제공하는 서버가 있다. InfoDID의 LDAP에 저장되는 사용자 정보는 사용자의 식별자인 DID, 사용자 정보 관리 시작 시각, 사용자 정보의 마지막 업데이트 시간, 사용자 정보의 유효 기간, 사용자의 성과 이름, 프로필 사진, 학력 목록 그리고 경력 등이 저장된다.

사용자 정보 관리 서비스인 InfoDID가 특정 사용자의 정보를 관리하고 있다면, 해당 사용자는 자신의 정보를 필요로 하는 서비스에 손쉽게 자신의 정보를 전달할 수 있으며 해당 정보를 사용한 이력을 InfoDID를 통해 관리할 수 있다. 그림 26과 같이 DID 사용자가 InfoDID를 이용해 은행 서비스에서 회원 가입 하는 간략한 과정은 다음과 같다.

[step1-2 in fig.26] 사용자는 DID 기반의 사용자 신원 인증 정보를 은행 서비스에 전달한다. 은행 서비스는 InfoDID에게 해당 사용자에 대한 신원 정보를 얻기 위하여, 받은 사용자 신원 인증과 자신의 신원 인증 정보를 InfoDID의 서버에게 보낸다.

[step3 in fig.26] InfoDID 서버는 은행 서비스로 받은 모든 정보를 이용하여 DID 기반으로 사용자에 대한 신원 인증, 은행 서비스에 대한 신원 인증을 진행하고 은행 서비스가 사용자 정보를 읽는 기능에 대한 사용자 승인 여부를 확인한다.

[step4-5 in fig.26] InfoDID 서버가 은행 서비스로부터 받은 정보에 대한 모든 검증이 완료되면, LDAP에 저장된 사용자 정보가 비정상적으로 변경되었는지 확인한다. 변경 여부에 대한 확인 완료되면 은행 서비스에게 사용자에 대한 정보를 발급했다는 기록을 InfoDID의 LDAP에 남기고 필요한 사용자 정보를 은행 서비스에게 전달한다.

이러한 민감한 사용자 정보를 관리하는 InfoDID 서비스는 서비스 실행에 대한 결함이 발생하여도 사용자에게 지속적인 서비스가 가능해야 하며 어떠한 상황에서도 결함에 대한 복구를 진행할 수 있어야 한다. 이에 따라, InfoDID 서비스와 같이 민감한 정보를 관리하는 블록체인 응용 서비스는 BR2K 기법을 적용할 필요가 있다.

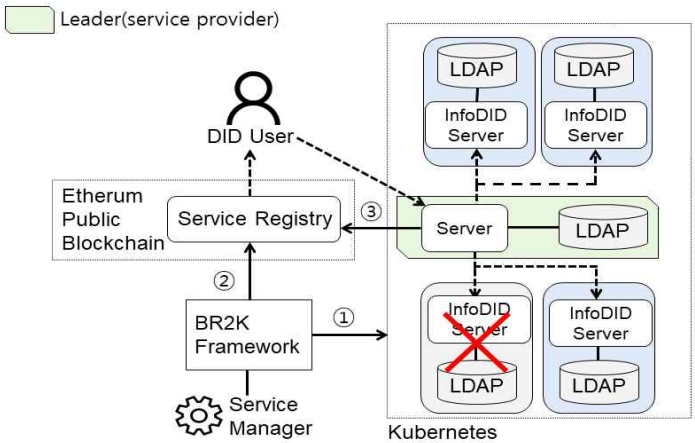

그림 27. 복제된 InfoDID 서비스

BR2K 프레임워크를 이용하여 BR2K 기법을 적용한 InfoDID 서비스는 그림 27과 같이 쿠버네티스 환경에서 복제 실행되어 각 노드에 서버와 LDAP가 실행되며, 그중 하나의 서버만이 BR2K 기법에 의해 사용자에게 서비스를 제공하는 리더 역할로 선출된다. 리더 역할을 가진 InfoDID 서버는 서비스 레지스트리에 자신의 위치를 현재 서비스 제공 위치로 업데이트하여 사용자에게 서비스를 제공하며 서비스 이름, 서비스 관리자와 같은 서비스 메타 정보 및 실행에 대한 복구 정보를 저장한다.

사용자는 서비스 레지스트리를 통해서 현재 InfoDID의 서비스를 제공하는 서버의 위치를 가져와, 자신의 정보를 저장(업데이트)하는 요청을 InfoDID의 서버에게 보낸다. 사용자로부터 요청을 받은 리더 서버는 해당 요청을 먼저 실행하여 해당 요청에 대한 오류 여부를 확인하고, 실행한 요청을 각 서버에게 복제하여 리더를 제외한 각 서버가 서로 같은 서비스 상태를 가지도록 한다. 만약 서비스를 제공하는 현재 리더 서버가 어떠한 이유로 중지되어도, BR2K 기법에 의하여 새로운 리더 서버가 선출되고 서비스 레지스트리를 통해 사용자에게 새로운 서비스 접속 정보를 제공할 수 있어 지속적으로 서비스를 제공할 수 있다. 또한, 쿠버네티스에서 실행이 중지된 InfoDID의 서버 및 LDAP는 실행에 대한 복구 정보를 이더리움 블록체인에서 실행되는 서비스 레지스트리에서 가져와 복구 작업을 수행할 수 있다.

6.2 테스트

BR2K 기법의 서비스 복제 실행의 유효성을 검증하기 위하여, 이 기법을 적용한 InfoDID를 테스트한다. 그림 28은 테스트 환경을 보여주며 12개의 가상 머신으로 구성된 쿠버네티스 환경에 BR2K 기법을 적용하여 하나의 ETCD와 InfoDID 서비스를 실행한다. BR2K 기법이 적용된 InfoDID 서비스에 요청을 보내는 사용자는 5명이며, 이 사용자들은 15개의 노드로 구성된 이더리움 블록체인에서 실행되는 서비스 레지스트리에서 서비스 제공 위치를 확인하여 사용자 요청을 보내게 된다.

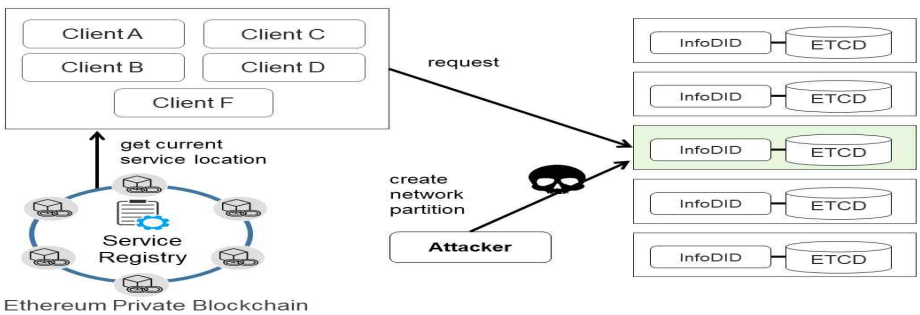

그림 28. 테스트 환경

그림 28에 Attacker는 30~120초 사이의 랜덤한 주기로 서비스 레지스트리를 통해 서비스를 제공하는 서버가 실행되는 노드를 찾아 네트워크 분할을 발생시키는 프로세스이다. 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 InfoDID 서버에게 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 InfoDID 서버는 요청에 대해서 처리하고 서비스 상태를 동기화한다. 이때, InfoDID 서버는 사용자 요청에 포함되어 있는 개인 정보를 InfoDID 서비스에 사용하는 로컬 데이터베이스인 LDAP에 저장한다. 이러한 작업이 완료되면, 사용자에게 해당 요청에 대한 성공 응답을 보낸다.
(3) 모든 사용자가 10,000번의 요청을 완료하면, 복제된 각 InfoDID 서버의 LDAP에 저장된 사용자의 개인 정보들이 서로 같은지, 누락된 부분이 없는지 검사하여 서비스 복제가 정확하게 이루어졌는지 확인한다.

표 5. 테스트 결과

	결과
Total time	about 83min
Frequency of network partition	30~120sec
Service downtime	about 16min
# of leader changes	67
# of *Ignore* states (Out of 50,000 requests)	10,193
# of *Success* states (Out of 50,000 requests)	39.805
# of *InProcess* states (Out of 50,000 requests)	2
State replication	success

표 5는 테스트에 대한 결과이다. 이 테스트에서 전체 테스트 시간 동안 네트워크 파티션으로 인한 서비스 장애 시간은 약 5분 1의 정도 비율로 발생하였고 1번 장애가 발생할 때마다 약 15초 간 서비스를 제공하지 못함을 확인하였다. 이는 BR2K 기법에서 새로운 서비스를 제공하는 리더에 대한 선출 시간과 이더리움 블록체인 노드의 합의로 인하여 새로운 접속 정보가 쓰이는 시간에 의존하는 걸로 확인되었다. 특히, 대부분 이더리움 블록체인의 합의 알고리즘으로 인하여 새로운 접속 정보가 다소 느리게 업데이트되었으며, 이러한 문제는 합의 속도가 이더리움 블록체인보다 빠른 블록체인을 선택하여 본 기법에 적용하면 서비스 장애 시간은 더욱 줄어들 거로 보인다.

또한, 이 테스트에서는 리더가 요청을 처리하는 중에 네트워크 분할이 발생하여도 복제 서비스 상태에 대한 동기화가 정확하게 이루어지는지 확인한다. 표 5에 있는 테스트 결과에서 볼 수 있듯이, 빈번한 리더의 변경과 요청 처리 중에 장애가 발생하여(table 5, # of InProcess states) 상태에 대한 복구 작업이 일어났음에도 처리에 성공한 사용자 요청의 개수와(table 5, # of Success states) 각 서버가 가지고 있는 사용자의 개인 정보의 수가 서로 일치함을 확인하였다. 이는 장애 상황에서도 상태 복제가 정확하게 이루어져 복제된 서비스 간의 일관성이 보장되고 지속적으로 서비스가 제공할 수 있음을 나타낸다.

7. 결론

본 논문에서는 블록체인 응용 서비스의 견고성을 지원하는 BR2K 기법을 제안하였다. 이를 위해 분산 환경에서 블록체인 응용 서비스를 복제하여 실행하는 기법을 개발하였고 이 기법이 적용된 복제 서비스가 전반적으로 실패할 때 신속한 복구를 지원하는 서비스 레지스트리를 개발하였다. 또한 제안된 기법의 실용성을 확보하기 위하여, 실제 개발 환경에서 해당 기법을 손쉽게 적용하기 위한 프레임워크 및 웹 모니터링 도구를 개발하였다. 이와 더불어, 이 기법을 시범적인 블록체인 응용 서비스인 사용자 정보 관리 서비스에 적용하고 이를 테스트하여 유효성을 확인하였다.

BR2K는 Raft 컨센서스 프로토콜 기반의 분산 저장소인 ETCD를 기반으로 개발된 체계적인 서비스 복제를 통하여, 네트워크 실패와 같은 장애 상황에서도 블록체인 응용 서비스가 사용자에게 서비스를 지속적으로 제공할 수 있도록 하였다. 또한, 쿠버네티스 배포에 대한 자원 명세 기법과 노드 라벨링 기반의 배포 방법을 제안하여, 해당 기법이 적용될 블록체인 응용 서비스가 분산 환경에서 신속하게 구성될 수 있도록 하였다. 이와 더불어, 이더리움 블록체인의 스마트 컨트랙트 기반으로 개발된 서비스 레지스트리를 통하여, 복구 정보에 대한 안전한 관리를 지원하였고 최신 접속 정보 제공 및 복구 요청 기능 등을 지원하여 재난 상황에 의하여 실패한 서비스에 대한 신속 재가동이 될 수 있도록 하였다.

본 논문에서는 실제 개발 환경에서 많이 사용되는 블록체인 개발 프레임워크인 트러플과 서버 프레임워크인 익스프레스 등을 확장하여 커맨드 라인 인터페이스 기반의 BR2K 프레임워크를 개발하였다. 또한, BR2K 기법이 적용된 블록체인 응용 서비스들을 모니터링하고 실패한 서비스에 대한 복구를 지원하는 웹 모니터링 도구인 BR2K Watch를 개발하였다. 이를 통해 실제 개발 환경에서 해당 기법을 손쉽게 적용할 수 있도록 하였고, 웹 브라우저 환경에서 서비스 결함 시에 대한 용이한 복구를 지원하도록 하여 본 기법의 실용성을 확보하였다. 추후 연구에서는 본 기법에 대한 처리 속도나 활용성을 높이기 위하여, 이더리움 블록체인뿐만 아니라 합의 속도가 빠른 클레이튼(Klaytn), 이오에스(EOS) 그리고 하이퍼레져 패브릭(Hyperledger fabric)과 같은 블록체인을 이용하여 해당 기법을 확장하고 추가적인 활용 도구들도 개발하여 실용성을 제고할 예정이다.

참 고 문 헌

[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.

[2] M Andoni, et al. "Blockchain technology in the energy sector: A systematic revie of challenges and opportunities" Renewable and Sustainable Energy Reviews, Vol. 100, pp. 143-174, Apr 2019.

[3] SN Khan, et al. "Blockchain Technology as a Support Infrastructure in E-Govern ent Evolution at Dubai Economic Department", Proceedings of the international, Ju 2019.

[4] Vida J. Morkunas, et al. "How blockchain technologies impact your business mod el, Business Horizons, Vol. 62, No. 3, pp. 295-306, May 2019.

[5] 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.

[6] S. Ølnes, A. Jansen, "Blockchain Technology as s Support Infrastructure in e-Gov rnment", Proceedings of the international conference on electronic government, p. 215-227, Sep 2017.

[7] H. Hou, "The application of blockchain technology in E-government in China", 20 7 26th International Conference on Computer Communication and Networks(ICC N), pp. 1-4, Jul 2017.

[8] S. Ølnes, "Beyond Bitcoin Enabling Smart Government Using Blockchain" Electronic Government, pp. 253-264, Sep 2016.

[9] Qin Wang, et al., "Non-Fungible Token (NFT): Overview, Evaluation, Opportunities and Challenges", SSRN Electronic Journal, 2021.

[10] K Wang, et al., "Securing data with blockchain and AI", IEEE Access, vol. 7, pp. 77981-77989, 2019.

[11] SK Singh, et, al., "Blockiotintelligence: A blockchain-enabled intelligent IoT architecture with artificial intelligence", Future Generation Computer Systems, 2019.

[12] Donguan Huang, et al. "Performance analysis of the raft consensus algorithm", I EE Transactions on Systems, Man, and Cybernetics: Systems, Vol. 50, No. 1, pp. 71-181, Jan 2020.

[13] ETCD, https://etcd.io

[14] Brewer, Eric "Kubernetes and the New Cloud", Proceedings of the 2018 Internation l Conference on Manaement of Data, pp.1, May 2018.

[15] P. McCorry, et al., "A smart contract for boardroom voting with maximum voter rivacy", International Conference on Financial Cryptography and Data Security, pp. 357-375, April 2017.
[16] M. Wohrer, U. Zdun, "Smart contracts: security patterns in the ethereum ecosyst m and solidity", Blockchain Oriented Software Engineering (IWBOSE) 2018 Internati nal Workshop on, pp. 2-8, Mar 2018.
[17] L. Lamport, "Paxos Made Simple", IEEE Transactions on Dependable and Secure omputing, pp. 1-52, Jan 2001.
[18] J. Z. Konczak, et al., "Recovery Algorithms for Paxos-based State Machine Replicat on", IEEE Transactions on Dependable and Secure Computing, pp. 1-1, July 2019.
[19] Docker, https://www.docker.com
[20] Truffle framework, https://www.trufflesuite.com
[21] Express framework, https://expressjs.com
[22] Vuejs, https://vuejs.org
[23] C Brunner, et al. "DID and VC:Untangling Decentralized Identifiers and Verifiable Credntials for the Web of Trust", In Proceedings of Preprint.ACM, pp. 6, 2020.
[24] 권민호, 이명준. “InfoDID: A robust user information management serivce based on Decentralized Identifiers”,한국컴퓨터정보학회지, 2021
[25] Drummond Reed, et al. "Decentralized Identifiers (DID) v1.0", W3C Working Draft, https://www.w3.org/TR/did-core

ABSTRACT

Techniques for robustness of blockchain application services

Blockchain is drawing attention from all over the world because of its technical characteristics that allow individuals and organizations to transparently store and manage information without the intervention of a central administrator. In particular, it has been rapidly developed and applied in areas where work history and data reliability are important, such as public infrastructure, medical care, real estate, and finance. However, in case the blockchain-based service fails to provide the service due to situations such as natural disaster, equipment failures, or cyber attacks, it takes a lot of time and money to restore it. Moreover, sometimes irrecoverable loss may occur. So, there needs a technique to support robustness of blockchain application services that ensures service continuity and quick restart.

In this paper, the BR2K technique and utilization tool developed to support the robustness of blockchain application services are described. The BR2K technique supports blockchain application services to continuously provide services to users through a recovery method that supports replication of services and quick restart for service failures. This technique replicates blockchain application services through replication of user requests using the etcd distributed storage, systematic processing of the requests, and distribution using Kubernetes to prepare for failures and ensure consistency of the replicated services. In addition, by presenting a service registry based on blockchain for the role of a disaster recovery center, information for accessing services is provided to users at any time. Also, backup information is reliably managed through the registry so that the service is promptly recovered even if the service is completely failed at the worst situation. In addition, we provide the associated tools developed to ensure the usefulness of the proposed technique, and the validity of the technique is shown by applying and testing the technique to a sample blockchain application service.

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