

2023년도

학사학위논문

상권분석을 위한 분산 빅데이터 서비스 시스템 구현

Implementation of a distributed big data
system for commercial analysis

2023년 11월 27일

순천향대학교 공과대학
컴퓨터공학과

강민서

상권분석을 위한 분산 빅데이터
서비스 시스템 구현

Implementation of a distributed big data
system for commercial analysis

지도교수 이 상 정

이 논문을 공학사학위 논문으로 제출함

2023년 11월 27일

순천향대학교 공과대학
컴퓨터공학과

강민서

강 민 서 의 공학사학위논문을 인준함

2023년 11월 27일

심 사 위 원 홍 인 식 인

심 사 위 원 이 상 정 인

순천향대학교 공과대학

컴퓨터공학과

초 록

상권은 재화와 서비스를 통해 고객을 유인하는 지역적 범위를 나타내며, 상권 분석은 해당 지역의 유동인구, 매출액, 업종, 연령대 등을 분석하여 적절한 업종을 결정하는 과정이다. 상권분석은 위험 최소화, 판매 예상량 추정, 시장 파악, 입지 전략 등을 위해 필요하며, 이로 인해 창업자들이 큰 관심을 가지고 있다. 현재 제공되는 상권분석 서비스에는 사용의 어려움과 정보 부족 등의 단점이 있어 이를 보완하기 위한 노력이 필요하다.

본 논문에서는 기존 서비스의 단점을 보완하고자 목표를 설정하며, 기존 서비스의 한계점을 지적한다. 먼저, 사용자 편의성 부족으로 인해 원하는 정보를 얻기 위해 여러 번의 클릭이 필요한 문제가 있다. 두 번째로, 창업 시 매출액에 영향을 미치는 요인 중 고객 동선과 교통 편의성에 대한 정보 부재가 있다. 이에 대응하기 위해 User Interface 개선과 교통 관련 정보 추가를 제안한다. 수평적 확장할 수 있는 분산 아키텍처를 제공하는 MongoDB를 사용하고 데이터 업로드 작업이 무겁기 때문에 사용자 입장에서는 즉각적인 반응과 병렬처리를 위해 파이썬 기반의 분산 비동기 작업 큐 라이브러리인 Celery를 도입했다. 또한 분산형 애플리케이션을 개발하기 위해 Docker를 사용하여 상권분석 빅데이터 서비스 분산 시스템을 구현한다.

주요어 : 상권분석, 분산 데이터베이스, 분산 시스템

ABSTRACT

Commercial districts represent the regional scope of attracting customers through goods and services, and commercial district analysis is the process of determining the appropriate business type by analyzing the floating population, sales, business type, and age group in the region. Commercial analysis is necessary for risk minimization, sales estimation, market identification, and location strategy, which is why founders are very interested in it. The currently provided commercial district analysis service has disadvantages such as difficulty in use and lack of information, so efforts are needed to compensate for this.

In this paper, goals are set to compensate for the shortcomings of existing services and limitations of existing services are pointed out. First, there is a problem that multiple clicks are required to obtain desired information due to the lack of user convenience. Second, among the factors affecting sales when starting a business, there is a lack of information on customer movement and transportation convenience. To counter this, we propose to improve the user interface and add traffic-related information. Because it uses MongoDB, which provides a distributed architecture that can be horizontally scaled, and data uploads are heavy, for users, Celery, a Python-based distributed asynchronous task queue library, is introduced for immediate response and parallel processing. In addition, to develop decentralized applications, Docker is used to implement a commercial analysis big data service distribution system.

Keywords: commercial district analysis, distributed database, distributed system

차 례

제 1 장 서 론	1
제 2 장 이론적 배경	3
2.1 MongoDB	3
2.2 Celery	4
2.3 Docker	5
제 3 장 분산 빅데이터 시스템 구현	7
3.1 분산 데이터베이스	7
3.1.1 분산 데이터베이스 구성	7
3.1.2 분산 데이터베이스 구현	10
3.2 분산 시스템	14
3.2.1 분산 시스템 구성	14
3.2.2 분산 시스템 구현	19
제 4 장 상권분석 서비스 구현	21
4.1 상권분석 서비스	22
4.1.1 파일 업로드 기능	22
4.1.2 상권분석 결과 조회 기능	23
4.2 데이터베이스 설계	24
4.3 Docker 네트워크 구성	25
제 5 장 구현환경 및 테스트	28
5.1 분산 데이터베이스 테스트	28

5.2 분산 시스템 테스트	30
5.3 User Interface 테스트	32
제 6 장 결론	35
참고문헌	36
감사의 글	37

그 립 차 례

[그림 1] Celery 동작	4
[그림 2] Docker 구조	5
[그림 3] Replica Set 구성	7
[그림 4] Sharded Cluster 구성	9
[그림 5] Docker에서의 분산 데이터베이스 구성	10
[그림 6] Config Server 생성	11
[그림 7] Shard Server1 생성	12
[그림 8] Mongos 생성	13
[그림 9] Celery적용 후 기능 분리	14
[그림 10] Docker에서의 분산시스템 구성	15
[그림 11] RabbitMQ 메시지 브로커 AMQP	16
[그림 12] RabbitMQ 웹 기반 관리 대시보드	17
[그림 13] Celery 모니터링 대시보드	18
[그림 14] Docker에서의 분산 시스템 구현	19
[그림 15] Celery concurrency 확인	20
[그림 16] 전체 시나리오	21
[그림 17] 파일 업로드 시퀀스 다이어그램	22
[그림 18] 상권분석 결과 조회 시퀀스 다이어그램	23
[그림 19] django-extensions를 사용하여 그린 Commercial data model	24
[그림 20] Docker에서의 Front-End 구성	25

[그림 21] Docker에서의 Back-End 구성	26
[그림 22] store.csv 데이터	28
[그림 23] commercial.csv, store.csv를 업로드	29
[그림 24] commercial.csv, store.csv를 업로드 결과	29
[그림 25] 분산시스템과의 실행시간 비교	31
[그림 26] 상권영역위에 마우스를 hover한 경우	33
[그림 27] 상권영역을 클릭한 경우	34

제 1 장 서 론

상권이란 개별점포가 재화와 서비스의 제공을 통하여 고객을 유인할 수 있는 지역적 범위 즉, 고객동원 지구를 의미하는 데 상업적 거래가 이루어지는 공간적 범위를 말한다[1]. 상권분석이란 지역이나 적합한 업종을 판단할 수 있도록 해당 지역의 유동 인구, 매출액, 업종, 연령대 분포 등을 분석한 것이다[1]. 상권분석은 위험률 최소화, 판매 예상량 추정, 시장 파악, 입지 전략 전개 등의 이유로 필요로 하다[1]. 이러한 필요성 때문에 창업하는 사람들이 상권분석에 대한 많은 관심을 가지게 되었고 상권분석에 대한 다양한 자료들과 상권분석에 도움이 되는 사이트들이 제공되고 있는 것을 확인할 수 있다.

본 논문에서는 현재 제공되고 있는 상권분석 서비스(<https://sg.sbiz.or.kr/godo/index.sg>)를 보완하는 것을 목적으로 한다. 현재 제공되고 있는 상권분석 서비스에는 다음과 같은 단점이 있다. 첫 번째는 소상공인시장진흥공단에서 제공하는 상권분석 서비스에서 원하는 상권의 정보를 얻기 위해 지역을 선택해도 팝업으로 광역시도, 시군구, 상권영역, 주요 상권을 다시 클릭해야 상권분석 결과를 얻을 수 있다는 단점이 있다. 그래서 본 논문에서는 사용자가 한 번의 클릭으로 원하는 정보를 얻을 수 있는 User Interface를 제공한다. 두 번째는 창업론에 따르면 입지 조사 시 매출액에 영향을 미치는 요인들에 고객의 동선으로 찾아오기 쉬운 곳, 차량 출입이 자유로운 곳이라고 한다[2]. 고객의 동선으로 찾아오기 쉽게 하려면 교통이 잘 되어있어야 하고 차량 출입이 자유롭기 위해서는 주차장이 필요하다. 근데 현재 많이 사용하는 상권분석 서비스는 이 부분에 대한 정보를 제공하지 않고 있다. 그래서 본 논문은 버스 정류장, 주차장, 지하철역 위치도 같이 제공한다. 마지막으로 본 논문은 한 지역, 한 국가에 대한 상권분석을 제공하

는 것이 아니라 상권분석에 필요한 데이터를 가지고 있는 어느 지역이나 제공하는 것을 목적으로 하므로 상권분석을 위한 분산 빅데이터 서비스 시스템을 설계 및 구현한다.

본 논문은 다음과 같은 구성을 가지고 있다. 2장에서는 상권분석을 위한 분산 빅데이터 서비스 시스템을 구현하는 데 사용한 기술 스택과 사용한 이유에 관해 서술하고 3장에서는 분산 데이터베이스, 분산 시스템의 구성 및 구현에 관해 서술한다. 4장에서는 상권분석 서비스의 기능과 데이터베이스 설계에 관해 서술하고 마지막으로 5장에서는 분산 데이터베이스와 분산 시스템이 제대로 작동하는지 테스트하고 사용자가 한 번의 클릭으로 교통수단, 주차장 데이터와 함께 원하는 상권의 정보를 얻을 수 있는지에 대해 테스트한 결과를 보여준다.

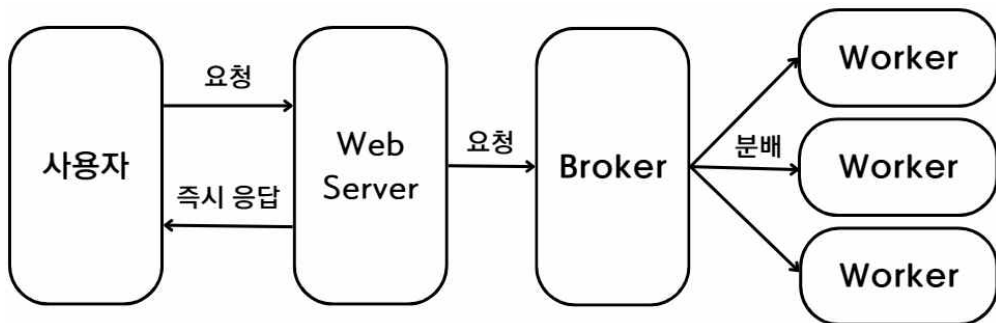
제 2 장 이론적 배경

2.1 MongoDB

MongoDB는 오픈소스 비 관계형 데이터베이스 관리시스템이다. 본 논문에서는 상권분석에 필요한 대용량의 데이터를 다루기 때문에 MongoDB의 특징들이 필요하다. MongoDB의 특징으로는 신뢰성, 확장성, 유연성, INDEX 지원이 있다. 각 특징에 대해 알아보자면 신뢰성은 서버에게 장애가 생기더라도 서비스가 계속 동작하도록 하는 것이다. MongoDB는 데이터를 쓰고 읽기 요청을 처리하는 주요 데이터베이스와 주요 데이터베이스를 복제한 두 개의 데이터베이스로 총 3개의 데이터베이스로 구성되어 있다. 이를 Replica Set이라고 하며 Replica Set 구성을 함으로써 장애가 발생해도 복제한 데이터베이스가 존재하기 때문에 서비스는 계속 동작할 수 있는 것이다. 그러다가 데이터가 하나의 Replica Set에 저장할 수 없을 정도로 양이 많아진다면 MongoDB는 수평 확장하여 데이터를 저장한다. 즉, 하드웨어를 더 좋은 것으로 바꾸는 것이 아닌 동일한 Replica Set을 더 추가하여 확장하는 것이다. 또한 MongoDB는 현재 테이블에 지정한 데이터가 아닌 다른 컬럼의 데이터가 들어오더라도 알아서 인식하여 컬럼을 만들고 저장해 주는 유연성을 가진다. 마지막으로 MongoDB는 다양한 조건으로 빠르게 데이터를 검색하고 조회할 수 있는 기능을 제공하고 Cluster에서 균등하게 분배하는 Index도 지원한다.

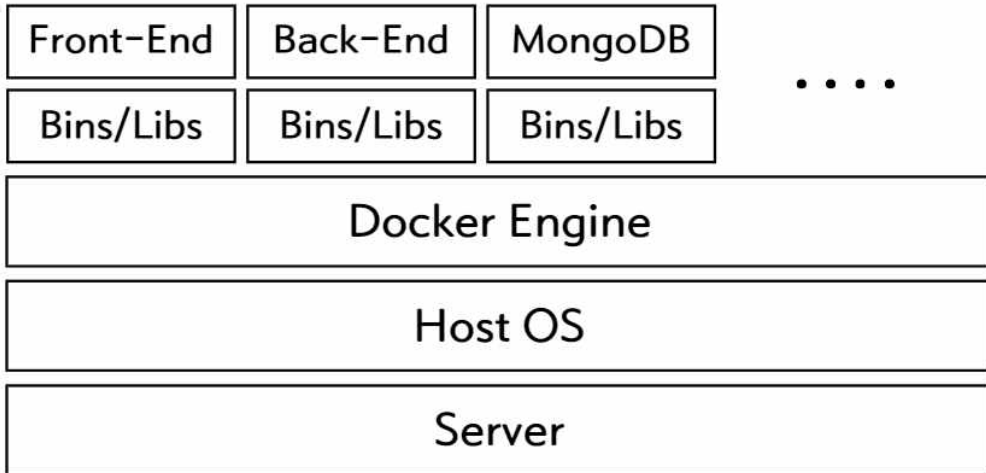
2.2 Celery

데이터 업로드 작업이 무겁기 때문에 사용자 입장에서는 즉각적인 반응 없이 지연이 발생하는 불편함이 생긴다. 이 불편을 해소하기 위해서 파이썬 기반의 분산 비동기 작업 큐 라이브러리인 Celery를 도입했다. Celery는 작업을 비동기로 처리하고 분산 환경에서 작업을 처리하는 방식을 가진다. 여기서 비동기는 병렬적으로 작업을 수행하는 것을 의미한다. Celery는 메시지를 통해 통신하며 일반적으로 Broker를 사용하여 Client와 Worker를 중재한다[3]. Broker에는 대표적으로 RabbitMQ와 Redis가 있다. Redis는 지속성이 중요하지 않고 약간의 손실을 견딜 수 있는 서비스에 적합하지만 RabbitMQ는 속도보다 지속성이 중요한 서비스에 적합한 특징을 가지고 있기 때문에 데이터가 중요한 상권 분석 서비스에는 RabbitMQ가 적합하다고 판단되어 본 논문에서는 RabbitMQ를 사용한다. Celery의 전체적인 동작은 [그림 1]과 같이 사용자가 Web Server에 요청하면 사용자는 즉시 응답 받을 수 있고, 웹 서버에서는 해당 요청을 Broker에게 전달한다. 그러면 Worker가 Broker에게 있는 요청을 받아서 처리하는 것이다.



[그림 1] Celery 동작

2.3 Docker



[그림 2] Docker 구조

Docker는 Container 기반의 오픈소스 가상화 플랫폼이다[4]. 여기서 Container는 표준화되고 실행 가능한 구성 요소로, 애플리케이션 소스 코드와 해당 코드를 임의의 환경에서 실행하는 데 필요한 운영체제 라이브러리 및 종속 항목이 조합된 것을 의미한다[5]. Docker의 핵심 개념은 이러한 Container를 활용하여 애플리케이션을 격리된 환경에서 실행하는 것이다. 기존에 사용하던 가상머신은 각각 독립된 OS와 응용 프로그램을 가지고 있어, 물리적 자원을 개별적으로 할당하여 사용하기 때문에 한정된 성능을 가지게 된다. Docker는 가상머신과 다르게 Host OS를 공유하여 독립된 OS를 가지지 않고 응용 프로그램만을 포함하여 가상머신보다 훨씬 가볍고 자원 소비를 최소화한다. 따라서 [그림 2]와 같이 Docker는

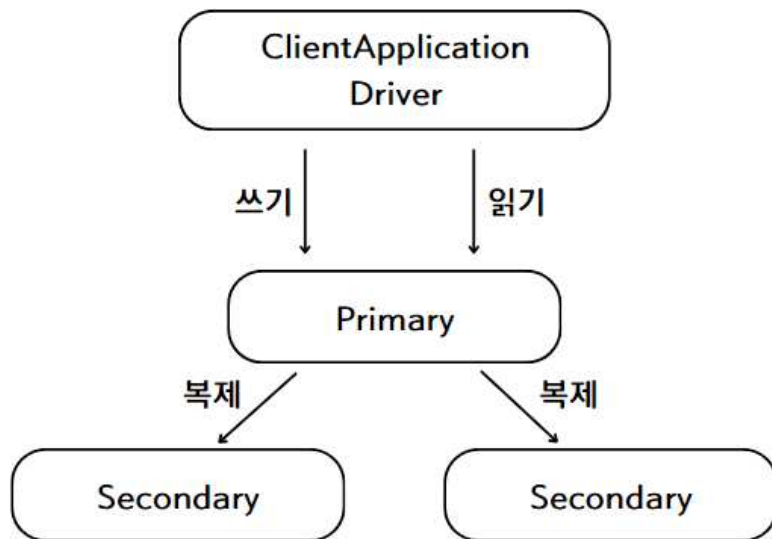
Host OS 위에 설치된 Docker Engine을 사용하여 Container를 생성, 실행, 관리한다. 이때 Container는 가상화 기술을 활용하여 독립적으로 실행되지만, Host OS를 공유한다.

본 논문에서는 분산 데이터베이스, 분산 시스템과 같은 분산형 애플리케이션을 개발하기 위해 Docker를 사용한다. 분산형 애플리케이션은 하나의 기능을 여러 개의 작은 부분들로 나눈 것을 의미한다. 이 부분에 대해서는 3장 분산 빅데이터 시스템 구현에서 더 자세하게 다룬다.

제 3 장 분산 빅데이터 시스템 구현

3.1 분산 데이터베이스

3.1.1 분산 데이터베이스 구성

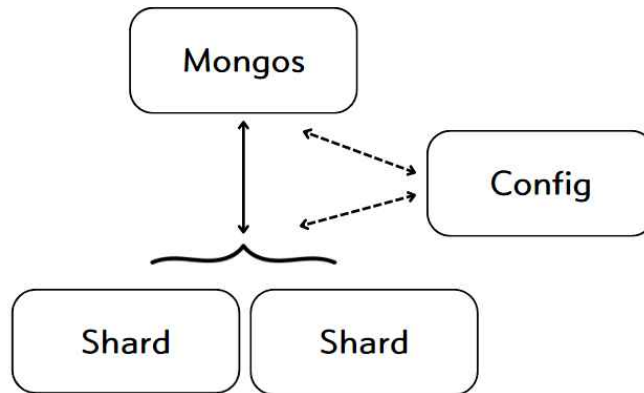


[그림 3] Replica Set 구성

2.1장에서 MongoDB에 관해 설명할 때, MongoDB는 주요 데이터베이스와 주요 데이터베이스를 복제한 두 개의 데이터베이스가 있다고 설명했다. 여기서 주요 데이터베이스를 Primary라고 부르고 주요 데이터베이스를 복제한 2개의 데이터베이스는 Secondary라고 부른다. 즉, Replica Set은 [그림 3]과같이

Primary와 Secondary로 구성되어 있다. Primary는 쓰기, 읽기 작업을 처리하고 Secondary는 동일한 데이터를 유지하기 위해서 Primary의 작업을 복제한다. 만약에 Primary에 장애가 생겨 사용할 수 없게 되었을 때, Secondary가 Primary와 동일한 데이터를 가지고 있기 때문에 Secondary 중 하나를 Primary의 역할을 수행하게 하여 서비스를 계속 동작할 수 있게 고가용성을 지원한다. 그 때문에 하드웨어 장애나 다른 장애 상황에서도 데이터베이스 시스템의 가용성을 유지할 수 있다.

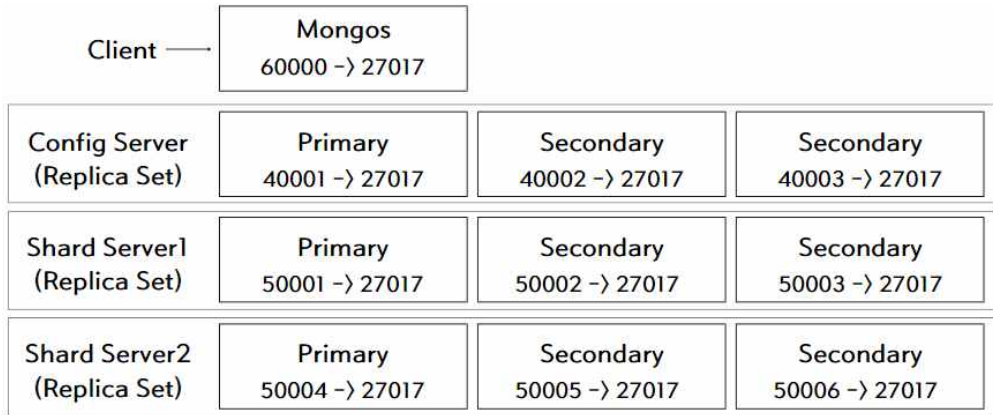
상권분석에 필요한 데이터에는 상권영역, 유동인구, 직장인구, 아파트 등이 포함된다. 이러한 데이터들은 한 지역에만이 아니라 여러 지역을 대상으로 하기 때문에 방대한 양의 데이터가 발생한다. 이로 인해 데이터 처리 속도가 느려지고 데이터를 감당할 수 없게 된다. 이런 상황에서 Replica Set을 추가하여 방대한 데이터를 효과적으로 관리할 수 있지 않을까라는 아이디어가 떠오르는데 여러개의 Replica Set을 사용하여 방대한 데이터를 관리할 수 있게 하는 것을 Sharded Cluster라고 한다.



[그림 4] Sharded Cluster 구성

Sharded Cluster는 [그림 4]에서 보듯이 Mongos, Config Server, Shard Server로 이루어져 있다. Mongos는 Sharded Cluster에서 중요한 역할을 하는데, Client가 직접 Shard Server에 연결하지 않고 Mongos를 통해 데이터베이스에 접근한다. Mongos는 Client의 요청을 받아들이고, 해당 요청을 적절한 Shard Server로 라우팅한다. 이러한 라우팅은 Shard Key 기반으로 이루어지며, 데이터를 어떤 Shard Server로 보낼지 결정한다. 여기서 Shard Key는 데이터를 분산하는데 사용되는 기준이고 만약 데이터베이스가 컴퓨터공학과 학생정보를 저장한다면 Shard Key는 학번이 될 수 있다. Config Server는 Cluster 내의 Shard Key 범위, Shard Server 위치 및 구성 정보 등의 메타데이터를 저장하는 역할을 한다. 또한, Config Server의 중요한 역할 중 하나는 Shard Key 범위를 기반으로 데이터를 어떻게 분산저장할지 결정하는 것이다. 마지막으로 Shard Server는 실제 데이터베이스 노드로 데이터를 저장하며 2개 이상이 필요하다. 또한 Config Server, Shard Server는 Replica Set 구성을 가지고 있다.

3.1.2 분산 데이터베이스 구현



[그림 5] Docker에서의 분산 데이터베이스 구성

본 논문에서는 [그림 5]와 같이 Sharded Cluster를 구현하기 위해 Mongos, Config Server, Shard Server1, Shard Server2로 구성되어 있고 Config Server, Shard Server1, Shard Server2는 Replica Set을 가지기 때문에 Docker에 MongoDB Image를 가진 10개의 노드를 생성해야 한다. 또한 이 10개의 노드는 내부에서 설정한 MongoDB 포트 27017로 접속할 수 있도록 포트 포워딩했다. 60000 포트를 통해 MongoDB에 접근할 수 있고 40001 포트를 통해 메타데이터를 저장하며 50001, 50004 포트를 통해 데이터를 분산하여 저장할 수 있다.

```
(base) hadoop@hadoop:~$ docker ps --format "table {{.ID}}\t{{.Image}}\t{{.Ports}}\t{{.Names}}" | grep "cfgsvr"
```

CONTAINER ID	IMAGE	PORTS	NAMES
77ce45ce29db	mongo	0.0.0.0:40003->27017/tcp, :::40003->27017/tcp	cfgsvr3
523887f54f49	mongo	0.0.0.0:40002->27017/tcp, :::40002->27017/tcp	cfgsvr2
adfbbf7c7f7a	mongo	0.0.0.0:40001->27017/tcp, :::40001->27017/tcp	cfgsvr1

```
(base) hadoop@hadoop:~$ mongo mongodb://220.69.209.126:40001
```

```
cfgrs:PRIMARY) rs.status().members.map(member => {
...   print('_id: ${member._id}, name: ${member.name}, stateStr: ${member.stateStr}');
... });
```

```
_id: 0, name: 220.69.209.126:40001, stateStr: PRIMARY
_id: 1, name: 220.69.209.126:40002, stateStr: SECONDARY
_id: 2, name: 220.69.209.126:40003, stateStr: SECONDARY
```

[그림 6] Config Server 생성

[그림 6]은 Config Server를 생성한 결과를 나타낸다. 이전에 설명한 대로 Config Server는 Replica Set을 필요로 한다. Replica Set을 만들기 위해서는 3개의 노드를 생성해야 하며, 각 노드를 Config Server 역할로 설정하고 Replica Set의 이름을 "cfgrs"로 정한다. 그리고 외부에서 40001, 40002, 40003으로 접속하면 내부적으로는 27017 포트로 접속하도록 포트포워딩을 설정한다. 이 설정이 제대로 완료되었는지 확인하기 위해 docker ps 명령어를 사용하였고 결과는 [그림 6]에서 볼 수 있다. 그 후, 생성한 3개의 노드를 연결해주어야 하는데 "mongodb://220.69.209.126:40001"에 접속하여 "rs.initiate" 함수를 사용하여, 생성한 3개의 노드가 Config Server의 구성원임을 알려준다. 이 설정이 제대로 완료되었는지는 "rs.status" 함수를 사용하여 확인할 수 있다. [그림 6]에 보이듯이 40001이 Primary, 40002와 40003이 Secondary로 제대로 연결된 것을 알 수 있다.

```
(base) hadoop@hadoop:~$ docker ps --format "table {{.ID}}\t{{.Image}}\t{{.Ports}}\t{{.Names}}" | grep
'shard1svr'

CONTAINER ID   IMAGE     PORTS                                     NAMES
5ca50bd2891f   mongo    0.0.0.0:50001->27017/tcp, :::50001->27017/tcp   shard1svr1
cd408d5b0972   mongo    0.0.0.0:50002->27017/tcp, :::50002->27017/tcp   shard1svr2
9b81b3f8e720   mongo    0.0.0.0:50003->27017/tcp, :::50003->27017/tcp   shard1svr3

(base) hadoop@hadoop:~$ mongo mongodb://220.69.209.126:50001

shard1rs:PRIMARY> rs.status().members.map(member => {
... print('_id: ${member._id}, name: ${member.name}, stateStr: ${member.stateStr}');
... });
_id: 0, name: 220.69.209.126:50001, stateStr: PRIMARY
_id: 1, name: 220.69.209.126:50002, stateStr: SECONDARY
_id: 2, name: 220.69.209.126:50003, stateStr: SECONDARY
```

[그림 7] Shard Server1 생성

[그림 7]은 Shard Server1을 생성한 결과를 나타낸다. Config Server처럼 Shard Server1도 Replica Set을 필요로 한다. Replica Set을 만들기 위해서는 3개의 노드를 생성해야 하며, 각 노드를 Shard Server 역할로 설정하고 Replica Set의 이름을 "shard1svr"로 정한다. 그리고 외부에서 50001, 50002, 50003으로 접속하면 내부적으로는 27017 포트로 접속하도록 포트포워딩을 설정한다. 이 설정이 제대로 완료되었는지 확인하기 위해 docker ps 명령어를 사용하였고 결과는 [그림 7]에서 볼 수 있다. 그 후, 생성한 3개의 노드를 연결해주어야 하는데 "mongodb://220.69.209.126:50001"에 접속하여 "rs.initiate" 함수를 사용하여, 생성한 3개의 노드가 Shard Server의 구성원임을 알려준다. 이 설정이 제대로 완료되었는지는 "rs.status" 함수를 사용하여 확인할 수 있다. [그림 7]에 보이듯이 50001이 Primary, 50002와 50003이 Secondary로 제대로 연결된 것을 알 수 있다. Shard Server2도 같은 방식으로 처리해준다.

```
(base) hadoop@hadoop:~$ docker ps --format "table {{.ID}}\t{{.Image}}\t{{.Ports}}\t{{.Names}}" | grep
"mongos"

CONTAINER ID   IMAGE     PORTS
b3601d56fe37   mongo    0.0.0.0:60000->27017/tcp, :::60000->27017/tcp   mongos

(base) hadoop@hadoop:~$ mongo mongodb://220.69.209.126:60000

mongos> sh.status()

shards:
[{"_id": "shard1rs", "host": "shard1rs/220.69.209.126:50001,220.69.209.126:50002,220.69.209.126:50003",
"state": 1, "topologyTime": Timestamp(1697540741, 2)}]

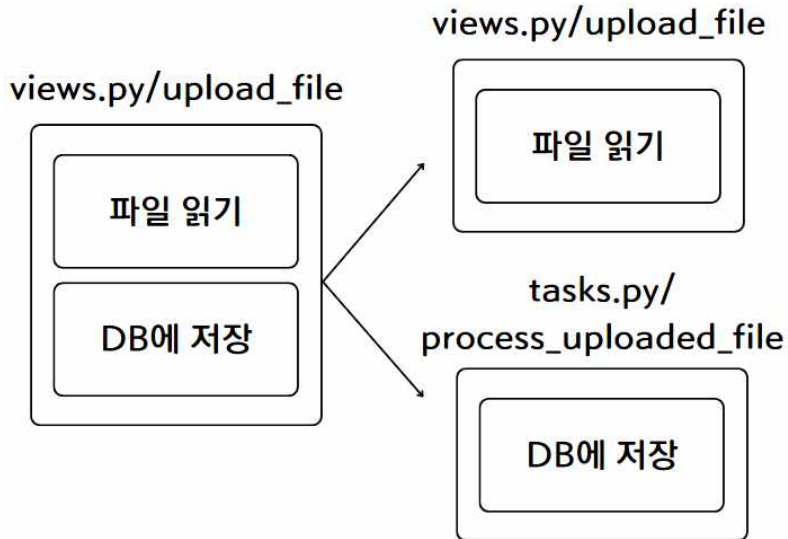
[{"_id": "shard2rs", "host": "shard2rs/220.69.209.126:50004,220.69.209.126:50005,220.69.209.126:50006",
"state": 1, "topologyTime": Timestamp(1697540797, 2)}]
```

[그림 8] Mongos 생성

[그림 8]은 Mongos를 생성한 결과를 나타낸다. Mongos는 Config Server와 Shard Server와는 다르게 Replica Set을 가지지 않기 때문에 1개의 노드를 생성해야 하고 docker ps 명령어를 사용하여 [그림 8]처럼 생성한 것을 확인할 수 있다. 위에서 생성한 Shard Server1, Shard Server2를 연결해주기 위해 "mongodb://220.69.209.126:60000"에 접속하여 "sh.addShard" 함수를 사용하여 Mongos에 Shard Server를 연결한다. 이 설정이 제대로 완료되었는지는 "rs.status" 함수를 사용하여 확인할 수 있다. [그림 8]에 보이듯이 Shards 목록에 Shard Server1과 Shard Server2가 제대로 연결된 것을 알 수 있다. 또한 Shard Server의 개수를 늘려서 추가하면 수평적 확장을 할 수 있다.

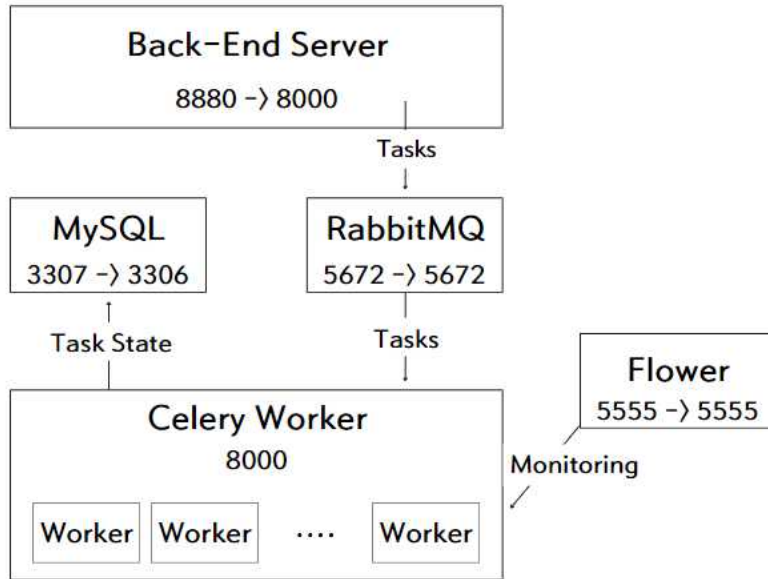
3.2 분산 시스템

3.2.1 분산 시스템 구성



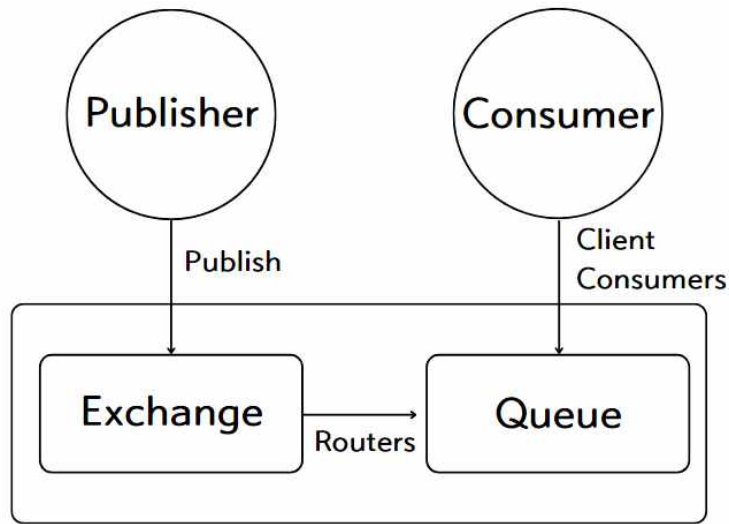
[그림 9] Celery적용 후 기능 분리

[그림 9]를 살펴보면 Celery를 도입하기 전에는 `views.py/upload_file`에 파일 읽기와 데이터 저장이 함께 이루어져 있는 것을 볼 수 있다. 그러나 이 중에서 데이터 저장 작업의 시간이 오래 걸려 작업이 끝날 때까지 사용자가 기다려야 하는 문제가 있다. 이에 따라 `views.py/upload_file`에서는 파일 읽기 작업만을 처리하고, 읽은 파일은 `tasks.py/process_uploaded_file`로 전달한다. 그리고 `tasks.py/process_uploaded_file`에서는 전달받은 파일의 데이터를 저장하는 무거운 작업을 비동기로 처리한다.



[그림 10] Docker에서의 분산시스템 구성

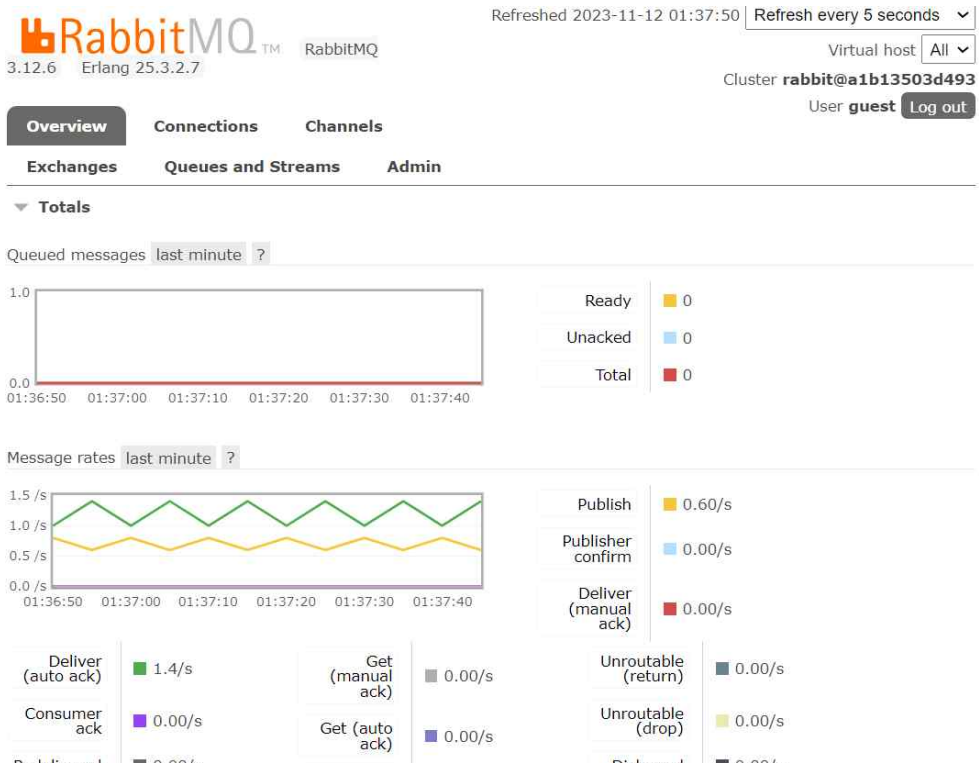
본 논문에는 [그림 10]과 같이 분산 시스템을 구축한다. Back-End Server에서 파일 데이터를 저장하는 기능이 발생하면 RabbitMQ를 통해 Celery Worker로 비동기 처리되도록 설계했다. 동시에 여러 작업이 들어올 경우, Celery Worker에는 여러 개의 Worker가 동시에 작업을 수행할 수 있어 처리 속도를 향상시킬 수 있다. Worker의 개수는 CPU 코어의 개수로 설정되어 있으며, 본 논문에서는 32개의 Worker를 사용했다. Flower는 Celery의 작업을 모니터링 한다. 5555 포트로 접속하면 어떤 작업이 실행되고 있고 이 작업이 정상적으로 완료되었는지 어떤 이유로 실패되었는지를 모니터링할 수 있다. MySQL은 Worker가 작업이 끝나면 작업의 성공 여부, 작업의 반환 값, 작업의 실행시간 및 타임스탬프 등의 작업 상태를 저장해야 하는데 이러한 데이터를 저장하도록 하였다.




[그림 11] RabbitMQ 메시지 브로커 AMQP

본 논문에서는 RabbitMQ에서 지원하는 AMQP 프로토콜을 사용한다. AMQP는 Advanced Message Queuing Protocol의 약자로 클라이언트 애플리케이션이 안정적으로 메시징 미들웨어 브로커와 통신할 수 있도록 하는 메시징 프로토콜이다. AMQP는 [그림 11]과같이 Publisher, Exchange, Queue, Consumer로 구성되어 있다. Publisher는 메시지를 생성하고 전송하여 Exchange에 Publish 한다. Exchange는 Publish 한 메시지를 Queue로 라우팅하는 역할을 한다. Queue는 1개 이상을 생성하여 사용할 수 있으며 전달받은 메시지를 저장하는 메시지 대기열이다. Consumer는 Queue를 구독하고 메시지를 가져와서 처리하는 역할을 한다. 여기서 Publisher, Consumer는 Celery가 된다. 본 논문에서 [그림 11]의 동작은 5672 포트를 통해 사용되고 15672 포트는 RabbitMQ에서 제공하는 웹 기반

관리 대시보드를 사용할 수 있다. 이 대시보드를 통해 Queue, Exchange, User 등을 확인하고 제어할 수 있다.



[그림 12] RabbitMQ 웹 기반 관리 대시보드


Flower

☰

Show 15 workers Search:

Worker	Status	Active	Processed	Failed	Succeeded	Retried	Load
celery@3e4c4c258fcf	Online	0	9	7	2	0	0.6
Total		0	9	7	2	0	

Showing 1 to 1 of 1 workers

Previous
1
Next

[그림 13] Celery 모니터링 대시보드

3.2.2 분산 시스템 구현

```
(base) hadoop@hadoop:~$ docker ps --format "table {{.Image}}\t{{.ID}}\t{{.Ports}}"
```

IMAGE	CONTAINER ID	PORTS
celery_worker	3e4c4c258fcf	8000/tcp
back_end	6b4899f42dc2	0.0.0.0:8880->8000/tcp
mher/flower	7cf9939c06c2	0.0.0.0:5555->5555/tcp
rabbitmq:3-management	a1b13503d493	0.0.0.0:5672->5672/tcp, 0.0.0.0:15672->15672/tcp
mysql:latest	13805f8e6adc	33060/tcp, 0.0.0.0:3307->3306/tcp

[그림 14] Docker에서의 분산 시스템 구현

3.2.1장에서 언급한 대로, 분산 시스템을 구축하기 위해서는 Back-End, Rabbit MQ, Flower, Celery Worker, 그리고 MySQL이 필요하다. 따라서 각각의 역할을 하는 노드를 생성하고 생성된 노드들을 확인하기 위해 `docker ps` 명령어를 사용하여 [그림 14]와 같이 노드들이 잘 생성되었는지 확인했다. 분산 시스템을 사용하기 위해서는 Back-End에 어떤 Broker로 작업을 전달하고, 작업의 상태를 어떤 데이터베이스에 저장할지 설정해 주어야 한다. 따라서 Back-End의 `Setting s.py` 파일에 다음과 같이 설정을 추가한다. “CELERY_BROKER_URL”에는 RabbitMQ의 AMQP 프로토콜을 사용하여 작업을 전달하겠다고 지정하고, “CELERY_RESULT_BACKEND”는 MySQL에 작업의 상태를 저장하겠다고 설정한다. 또한 Celery Worker는 RabbitMQ의 작업을 가져오고 Flower는 RabbitMQ를 모니터링하기 때문에 이 두 노드에도 Broker를 RabbitMQ의 AMQP 프로토콜을 사용한다고 지정해야 한다.

```

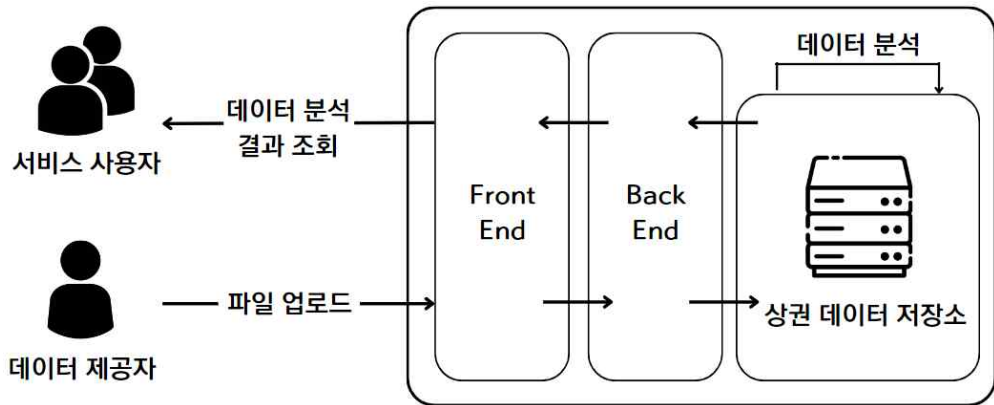
----- celery@3e4c4c258fcf v5.3.4 (emerald-rush)
-- ***** -----
-- ***** ----- Linux-5.15.0-86-generic-x86_64-with-glibc2.31 2023-11-12 01:24:20
-- *** --- * ---
-- ** ----- [config]
-- ** ----- .> app:      mysite:0x7f7896eec490
-- ** ----- .> transport:  amqp://guest:**@rabbitmq:5672//
-- ** ----- .> results:   mysql://root:**@220.69.209.126:3307/mycelerydb
-- *** --- * --- .> concurrency: 32 (prefork)
-- ***** ----- .> task events: OFF (enable -E to monitor tasks in this worker)
-- ***** -----
----- [queues]
      .> celery      exchange=celery(direct) key=celery

```

[그림 15] Celery Concurrency 확인

[그림 15]를 보면 “concurrency: 32”라고 적힌 것을 볼 수 있다. Concurrency는 Celery Worker가 동시에 처리할 수 있는 작업의 수를 나타내고 작업자가 동시에 몇 개의 작업을 처리할지를 조절하는 중요한 설정이다. Concurrency 설정을 통해 시스템의 자원을 효율적으로 활용하고, 작업 처리의 병렬성을 조절하여 성능을 최적화할 수 있다. 즉, 이 설정은 분산 시스템에서 여러 작업을 동시에 처리함으로써 전반적인 처리 속도와 성능을 향상하는 데 기여한다. [그림 15]에서도 나와 있듯이 본 논문에서는 32개의 Worker를 사용한다.

제 4 장 상권분석 서비스 구현

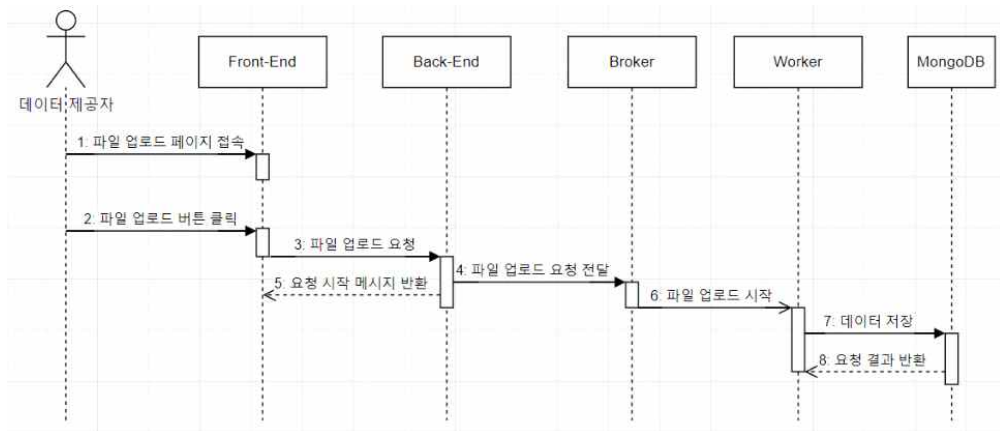


[그림 16] 전체 시나리오

상권분석 서비스는 데이터 제공자로부터 전달받은 상권 데이터 분석 결과를 여러 서비스 사용자에게 제공하는 서비스다. 전체 시나리오는 다음과 같다. 데이터 제공자가 상권분석에 필요한 파일을 Front-End를 통해 업로드하면 해당 파일은 Back-End로 전달되고 전달된 파일은 처리되어 데이터베이스에 저장된다. 서비스 사용자는 Front-End를 통해 원하는 상권영역을 선택하면 Back-End로 데이터 요청을 전달한다. Back-End는 데이터베이스에 저장된 데이터 분석한 결과를 응답으로 전달해 주고 Front-End는 이 데이터를 시각화하여 사용자에게 제공한다. 본 논문에서는 [그림 16]과 같이 사용자와 상호작용하여 요청을 전달하는 Front-End와 데이터 저장 및 분석하는 기능을 하는 Back-End와 데이터베이스가 필요하다.

4.1 상권분석 서비스

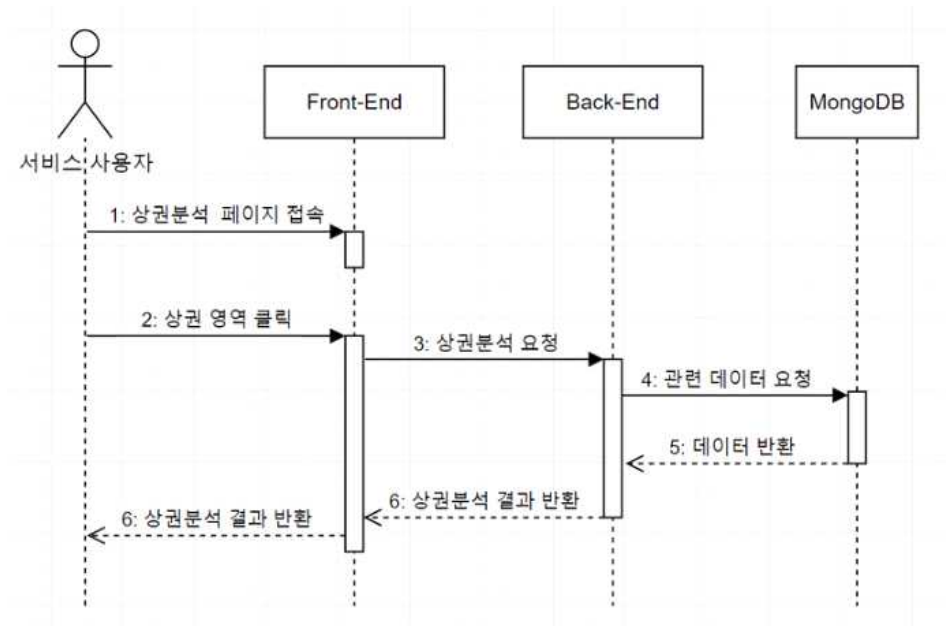
4.1.1 파일 업로드 기능



[그림 17] 파일 업로드 시퀀스 다이어그램

[그림 17]은 데이터 제공자가 파일 업로드 기능을 사용했을 때 발생하는 상호 작용 메시지를 시간의 흐름에 따라 그린 다이어그램이다. 데이터 제공자가 파일 업로드 페이지에 접속하면 데이터 제공자는 업로드하고자 하는 파일이 어디 지역 데이터인지 국가명과 도시명을 입력해 주고 데이터 종류에 맞게 파일을 선택해 준다. 제출 버튼을 누르면 Front-End에서 Back-End로 파일 업로드 요청을 보낸다. Back-End는 파일 업로드를 비동기로 처리하기 때문에 Broker로 전달하고 Front-End로는 작업이 시작되었다는 메시지를 반환한다. 작업을 전달받은 Broker는 32개의 Worker 중 하나를 파일 업로드 작업을 시작하게 하고 Worker는 파일에 작성된 데이터를 MongoDB에 저장한다.

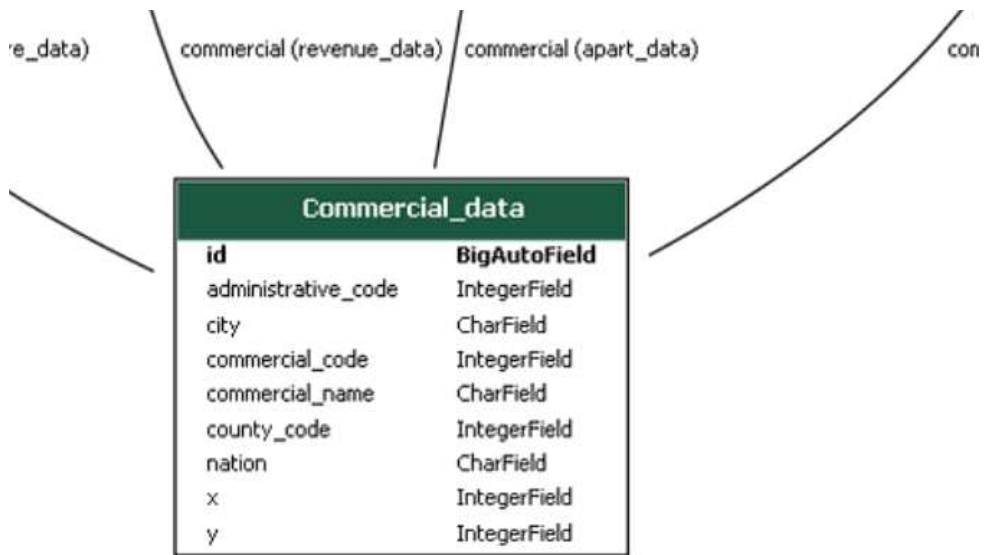
4.1.2 상권분석 결과 조회 기능



[그림 18] 상권분석 결과 조회 시퀀스 다이어그램

[그림 18]는 서비스 사용자가 상권분석 결과 조회 기능을 사용했을 때 발생하는 상호작용 메시지를 시간의 흐름에 따라 그린 다이어그램이다. 서비스 사용자가 상권분석 페이지에 접속하면 서비스 사용자는 조회하고자 하는 상권영역을 선택한다. Front-End는 서비스 사용자가 선택한 상권영역의 상권 코드와 상권 명을 함께 Back-End로 전달하여 상권분석을 요청한다. Back-End는 전달받은 상권 코드와 상권 명을 통해 MongoDB에 관련 데이터를 요청하여 전달받고 전달받은 데이터를 분석하여 Front-End로 분석한 결과를 전달한다. Front-End는 전달받은 데이터를 시각화하여 서비스 사용자에게 제공한다.

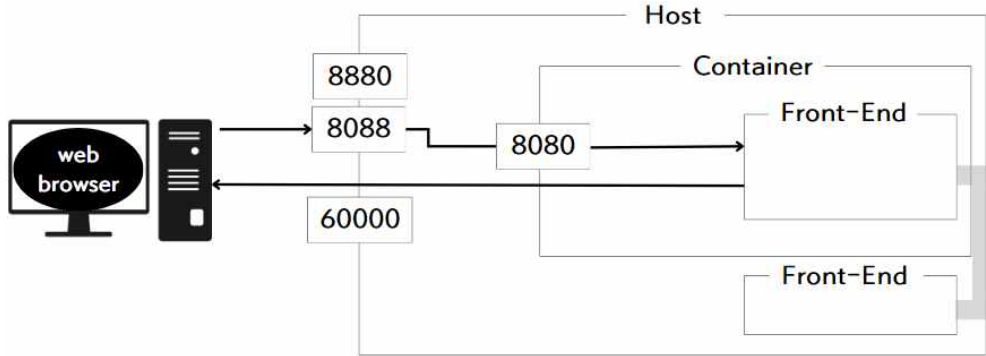
4.2 데이터베이스 설계



[그림 19] django-extensions를 사용하여 그린 Commercial data model

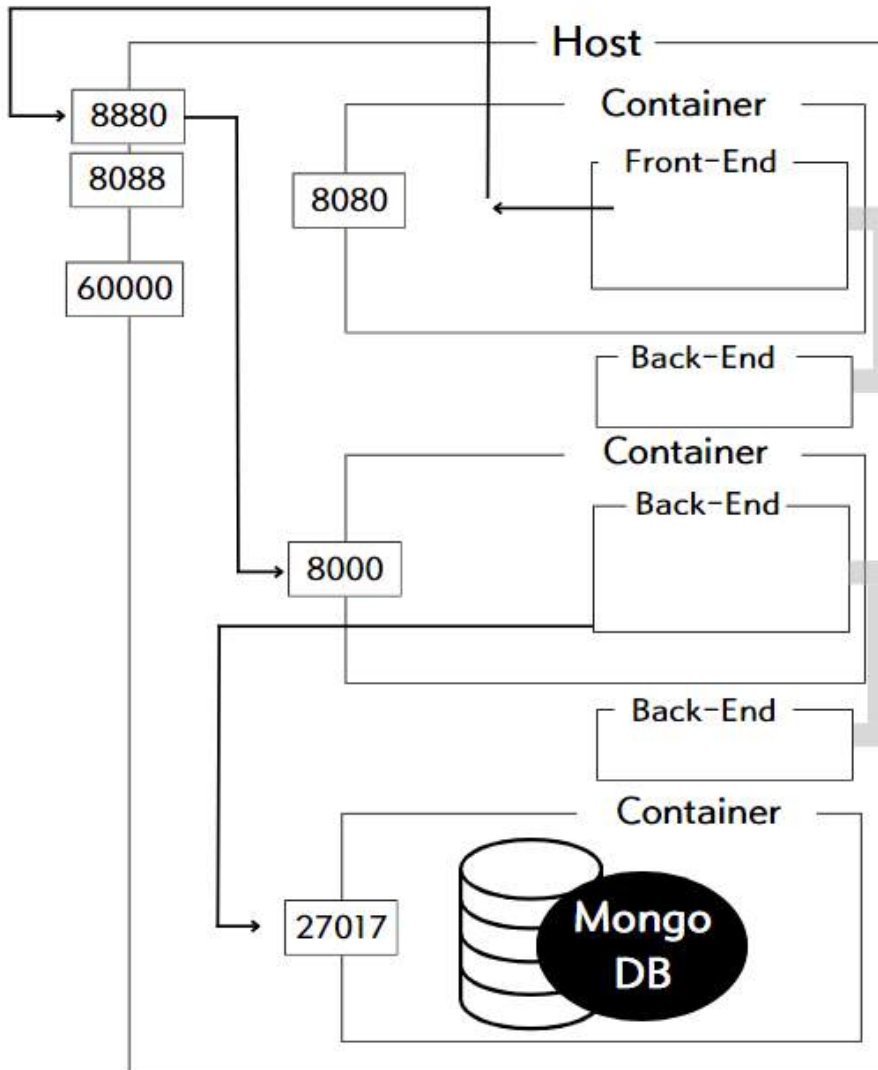
[그림 19]에서 Commercial data model은 행정동 코드, 국가명, 도시명, 상권 코드, 상권 명, 시군구 코드, 중심위치(x, y)가 존재한다. 상권 코드, 상권 명을 가지고 아파트, 유동 인구 등 관련 데이터를 찾기 위해서는 아파트, 유동 인구 등의 데이터들도 상권 코드, 상권 명 데이터를 저장해야 한다. 하지만 이러한 방식은 Commercial data가 바뀌면 다른 아파트, 유동 인구 등의 데이터도 전부 찾아 변경해 줘야 하여 무결성 문제가 발생할 수 있을 뿐만이 아니라 똑같은 데이터를 저장함으로써 데이터베이스 용량이 낭비될 수 있다. 그래서 상권분석에 필요한 아파트, 유동인구 등의 데이터들은 Commercial data를 외래키로 가지고 있다.

4.3 Docker 네트워크 구성



[그림 20] Docker에서의 Front-End 구성

[그림 20]에서는 Docker에서의 Front-End 구성을 보여준다. Host와 Container는 각각 독립된 실행 환경이기 때문에 각자 독립적인 포트와 Front-End 작업 폴더를 가지고 있다. Front-End Container를 실행할 때 “python manage.py runserver 0.0.0.0:8080” 명령어를 실행하여 Container는 8080 포트를 가진다. 사용자는 Host를 통해서 Container에 접근하기 때문에 포트 포워딩이 필요하다. 이를 위해 “docker run -d -p 8088:8080 front_end” 명령어를 사용하여 Host의 8088 포트를 Container의 8080 포트로 매핑해준다. 이렇게 함으로써 외부에서 Host의 8088 포트로 들어오는 요청은 Container의 8080 포트로 전달되어 사용자가 Front-End에 접근할 수 있다. [그림 20]을 보면 Host의 Front-End와 Container의 Front-End가 연결된 것을 확인할 수 있다. 이 연결은 Host에서 Front-End 작업 폴더를 수정하고 수정한 내용을 Container에 적용하게 함으로써 Container에 문제가 생겨도 더 안전하게 개발하기 위함이다.



[그림 21] Docker에서의 Back-End 구성

[그림 21]에서는 Docker에서의 Back-End 구성을 보여준다. Back-End Container를 실행할 때 “python manage.py runserver 0.0.0.0:8000” 명령어가 실행되도록 했기 때문에 Container는 8000 포트를 가진다. Front-End는 Host를 통해서 Back-End에 접근하기 때문에 포트 포워딩을 해야 한다. “docker run -d -p 8880:8000 back_end” 명령어를 통해 Host의 8880 포트를 Container의 8000 포트로 매핑해준다. 이렇게 함으로써 외부에서 Host의 8880 포트로 들어오는 요청은 Container 8000 포트로 들어가 사용자가 Back-End에 접근할 수 있다. 또한 Back-End는 MongoDB에 접근하여 읽기, 쓰기 작업을 한다. 단순히 포트 포워딩을 해줬다면 Back-End에서 60000 포트로 요청을 보내고 27017 포트에 접근할 수 있는데 [그림 21]에서 Back-End가 60000 포트가 아닌 27017 포트 요청을 보내는 것을 볼 수 있다. 이를 가능하게 하는 것을 브릿지 네트워크라고 한다. 브릿지 네트워크는 여러 개의 컨테이너를 동일한 네트워크에 연결하여 컨테이너 간에 내부 포트로 통신하게 해준다. 본 논문에서는 mynetwork라는 브릿지 네트워크를 생성하여 Back-End와 외부 포트로 통신하지 않아도 되는 MongoDB, RabbitMQ, Celery Worker를 mynetwork에 연결하여 내부 포트로 통신하게 했다.

제 5 장 구현환경 및 테스트

5.1 분산 데이터베이스 테스트

분산 데이터베이스에 데이터가 제대로 저장되는 지 테스트하고 4.2장에서 상권분석에 필요한 데이터들은 Commercial data를 외래키로 가진다고 했는데 실제로 데이터가 잘 저장되는 지 테스트한다. MongoDB는 7.0.2 버전을 사용했으며 [그림 23]처럼 commercial.csv와 store.csv를 업로드한다고 가정한다. [그림 22]는 store.csv 데이터로 상권코드는 “1001496”, 상권 명은 “강남 마이스 관광특구”이다. 즉, [그림 22]가 저장될 때, 상권코드는 “1001496”, 상권 명은 “강남 마이스 관광특구”인 Commercial data를 외래키로 가져야 한다. [그림 24]는 commercial.csv와 store.csv가 MongoDB에 저장된 상태다. commercial data를 보면 상권코드는 “1001496”, 상권 명은 “강남 마이스 관광특구”인 데이터는 id가 3341인 것을 볼 수 있고 store data도 commercial_id가 3341인 것을 보아 제대로 저장된 것을 확인 할 수 있다.

commercial_id	commercial_name	service_code	service_name	store_num	similar_store_num	opening_rate
1001496	강남 마이스 관광특구	CS300043	전자상거래	6	6	0

[그림 22] store.csv 데이터

파일 업로드

korea	seoul
-------	-------

데이터 종류	파일 올리기
상권영역(필수)	<input type="button" value="파일 선택"/> commercial.csv
점포	<input type="button" value="파일 선택"/> store.csv
추정매출	<input type="button" value="파일 선택"/> 선택된 파일 없음

[그림 23] commercial.csv, store.csv를 업로드

market.main_commercial_data

Documents

Aggregations

Schema

Indexes

Filter

Type a query: { field: 'value' }

ADD DATA

EXPORT DATA

```

_id: ObjectId('65454be795d463e0e21fabe4')
id: 3341
nation: "korea"
city: "seoul"
commercial_code: 1001496
commercial_name: "강남 마이스 관광특구"
x: 205310
y: 445727
county_code: 11680
administrative_code: 11680580

```

market.main_store_data

Documents

Aggregations

Schema

Indexes

Filter

Type a query: { field: 'value' }

ADD DATA

EXPORT DATA

```

_id: ObjectId('65454be795d463e0e21fabe6')
id: 74992
commercial_id: 3341
year: 2022
quarter: 4
service_code: "CS300043"
service_name: "전자상거래업"
store_num: 6
similar_store_num: 6
opening_rate: 0
opening_store_num: 0
closure_rate: 0
closure_store_num: 0
franchisee_store_num: 0

```

[그림 24] commercial.csv, store.csv를 업로드 결과

5.2 분산 시스템 테스트

분산 시스템을 테스트하기 위해서 3개의 파일을 하나의 작업으로 만들어 순차적으로 저장하는 경우와 3개의 파일을 3개의 작업으로 나누어 병렬로 저장하는 경우의 실행시간을 비교한다. 실행시간은 Flower의 Runtime 값을 사용했고 첫 번째 경우와 두 번째 경우 동일하게 상권영역, 점포, 추정 매출 파일을 사용했다. 첫 번째 경우에는 총 3개의 파일을 순차적으로 업로드 해야 하므로 작업이 끝나는 데까지 시간이 걸릴 것이다. 반면에 두 번째 경우에는 3개의 파일을 병렬로 처리하기 때문에 분산 시스템이 제대로 작동한다면 첫 번째 경우보다 더 빨리 작업이 끝날 것이다.

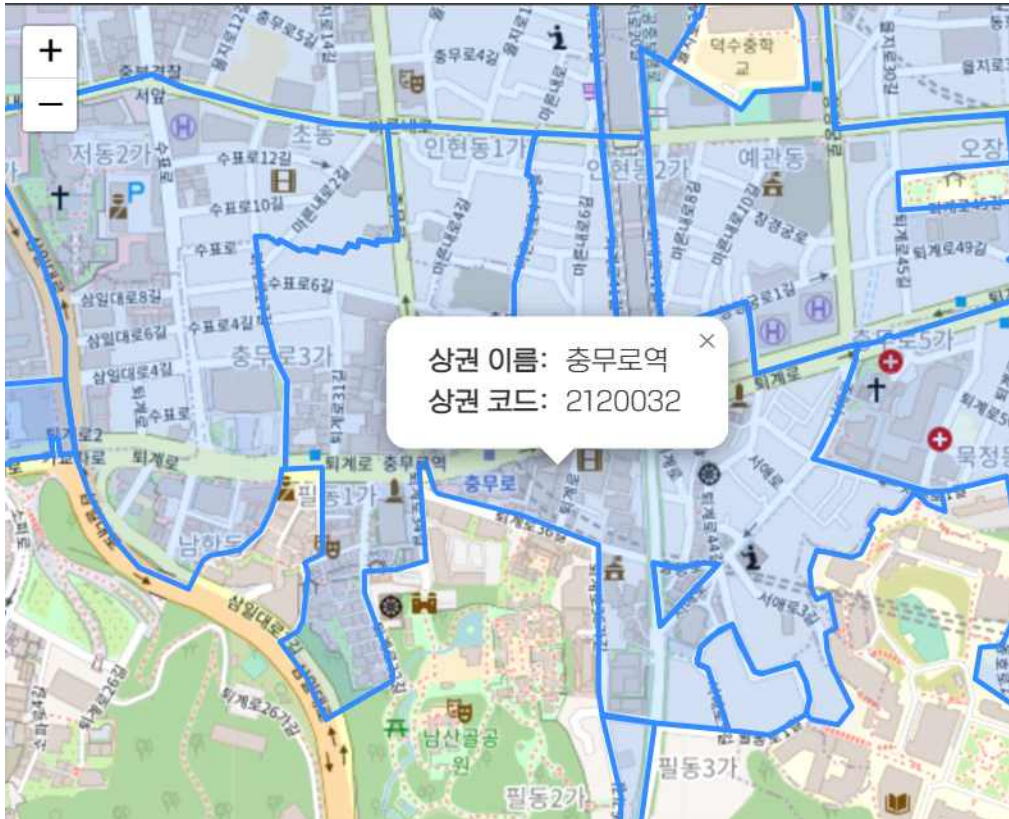
결과는 [그림 25]와 같다. [그림 25]에서 가장 왼쪽에 있는 막대는 첫 번째 경우 실행시간이고 그다음부터는 두 번째 경우의 상권영역, 점포, 추정 매출 실행시간을 나타낸다. 첫 번째 경우에는 3개의 파일을 저장하는 데 3188.03초가 걸렸고 두 번째 경우에는 상권영역 파일은 19.63초, 점포 파일은 1851.39초, 추정 매출 파일은 1523.7초가 걸려서 3개의 파일 모두 작업이 끝날 때까지의 실행시간은 가장 오래 걸린 점포 파일의 1851.39초이다. 이 결과를 보듯이 분산시스템을 사용함으로써 서비스의 속도를 향상했다는 것을 알 수 있다.



[그림 25] 분산시스템과의 실행시간 비교

5.3 User Interface 테스트

1장에서 사용자가 한 번의 클릭으로 원하는 정보를 얻을 수 있는 User Interface를 제공한다고 했다. 그래서 지도에 상권영역 공간데이터를 사용하여 공간을 구분해주고 사용자가 상권영역 위로 마우스를 hover하면 [그림 26]과 같이 어떤 상권인지 알려주는 popup이 나타난다. 또한 사용자가 원하는 상권영역 위로 마우스를 클릭하면 마우스가 클릭된 상권영역의 상권코드와 상권 명을 Back-End로 전달해 분석결과를 응답받고 분석결과를 [그림 27]처럼 시각화하여 사용자에게 제공한다. 왼쪽 하얀색 부분은 매출, 인구, 주거, 학원의 분석결과를 제공하고 1장에서 본 논문은 버스 정류장, 주차장, 지하철역 위치도 같이 제공한다고 했었는데 [그림 27]에서 오른쪽 지도에서 빨간색 마커가 버스 정류장, 주차장, 지하철역 위치를 가리킨다.



[그림 26] 상권영역위에 마우스를 hover한 경우



[그림 27] 상권영역을 클릭한 경우

상기와 같이 분산 데이터베이스와 분산 시스템을 구현하여 분산 저장과 처리를 통해 대규모 데이터에 대한 처리 능력과 User Interface의 유효성을 확인할 수 있었다.

제 6 장 결론

본 논문에서는 상권분석을 위한 분산 빅데이터 서비스 시스템을 구현했으며 사용자가 한 번의 클릭으로 원하는 정보를 얻을 수 있는 User Interface를 제공했고 버스 정류장, 주차장, 지하철역 위치도 같이 제공한다. 수평적 확장할 수 있는 분산 아키텍처를 제공하는 MongoDB를 사용하여 데이터를 효율적으로 분산 관리하고 확장할 수 있으며 분산 저장과 처리를 통해 대규모 데이터에 대한 처리 능력을 향상했다. 파이썬 기반의 분산 비동기 작업 큐 라이브러리인 Celery를 도입하여 사용자 입장에서 즉각적인 반응이 오지 않고 지연을 일으키는 작업을 비동기로 처리하여 불편을 해소했고 병렬처리를 하여 여러 작업을 동시에 처리하여 속도를 향상했다. 또한 분산형 애플리케이션을 개발하기 위해 Docker를 사용하여 고가용성을 제공했고 필요에 따라 시스템을 확장하여 늘어난 서비스 요청량에 대응할 수 있다.

향후 연구에서는 요청된 작업을 하나의 작업으로 보고 처리하지 않고 요청된 작업을 나눠서 처리하여 처리 속도를 더 향상하고 데이터 제공자가 데이터를 업로드한 뒤에 데이터 제공자가 필수 데이터를 빼먹지 않았는지 확인하는 기능을 넣어 생겨날 수 있는 오류를 예방할 방안을 제시한다.

참 고 문 헌

- [1] 채규옥(2017), 상권분석 및 입지전성, 2023.07.04,
<http://contents2.kocw.or.kr/KOCW/document/2017/wonkwang/chaekyuok1/5.pdf>
- [2] 최용석(2018), 외식사업창업론 상권조사와 입지분석, 2023.07.04,
<http://contents2.kocw.or.kr/KOCW/document/2018/seowon/choiyongseok0128/6.pdf>
- [3] Celery Docs, Celery 소개, 2023.09.20,
<https://docs.celeryq.dev/en/stable/getting-started/introduction.html>
- [4] subicura(2017), 초보를 위한 도커 안내서 - 도커란 무엇인가?, 2023.07.11,
<https://subicura.com/2017/01/19/docker-guide-for-beginners-1.html>
- [5] IBM 뉴스레터, Docker란, 2023.10.25,
<https://www.ibm.com/kr-ko/topics/docker>
- [6] 인파(2021), 몽고디비 특징 & 비교 & 구조 (NoSQL), 2023.02.18,
<https://inpa.tistory.com/entry/MONGO-%F0%9F%93%9A-%EB%AA%BD%EA%B3%A0%EB%94%94%EB%B9%84-%ED%8A%B9%EC%A7%95-%EB%B9%84%EA%B5%90-%EA%B5%AC%EC%A1%B0-NoSQL>
- [7] MongoDB Docs, Replication, 2023.02.18.,
<https://www.mongodb.com/docs/manual/replication/>
- [8] MongoDB Docs, Sharding, 2023.02.18,
<https://www.mongodb.com/docs/manual/sharding/>
- [9] Celery Docs, concurrency, 2023.09.20,
<https://docs.celeryq.dev/en/stable/userguide/workers.html#concurrency>

[10] 조은우(2019), RabbitMQ 동작 이해하기, 2023.09.20,

<https://jonnung.dev/rabbitmq/2019/02/06/about-amqp-implementation-of-rabbitmq/#gsc.tab=0>

감 사 의 글

대학교 생활을 하며 힘든 일이 있거나 아플 때, 인천에서 순천향대학교까지 바로 달려와 주시던 부모님께 먼저 감사 인사드립니다. 그리고 2021년 CS 연구실에 들어가 적응을 못할 때, 먼저 손을 내밀어 주신 김재진 선배와 마음씨가 이쁘시고 제가 도움을 요청하면 기꺼이 도움을 주셨던 민새미 선배 그리고 묵묵하게 자리를 지키고 공부 분위기를 만들어 준 랩장 송희령 선배, 지금도 배울 점이 많은 이인재 선배께 감사 인사드립니다. 2022년부터 지금까지 함께 했던 CS 연구실 부원들도 모두 감사드립니다. 연구실 부원들과 친해질 수 있는 계기를 만들어준 든든한 랩장 신주용 선배와 늘 제 옆에서 응원해 주던 귀여운 홍지민 후배, 같이 보드게임 해주던 김어진 후배, 곁에 있으면 편하고 힘든 일이 있으면 도와주던 윤준식 선배, 1학년 때부터 함께하고 정도 많이 들고 도움도 많이 받았던 나의 정신적 지주 이인규 동기에게도 감사 인사드립니다. 대학 생활을 함께하며 같이 고생했던 이유리, 김채린과 감정적으로 변하면 이성적으로 판단할 수 있게 도와주던 박세진과 성지현, 늘 옆에서 묵묵하게 응원해 주던 황지상 후배에게 감사드립니다.

마지막으로 이 논문을 작성하는데 매주 세미나를 통해 도움을 주신 이상정 교수님과 더 나은 논문을 작성할 수 있도록 도움을 주신 홍인식 교수님 그리고 4년 동안 수업을 통해 다양한 지식을 나눠주셨던 천인국 교수님, 하상호 교수님, 남윤영 교수님, 이해각 교수님께 감사 인사드립니다. 4년 동안 정말 많은 경험을 하고, 많은 것을 깨달았습니다. 여기서 받았던 모든 도움을 잊지 않고 살아가겠습니다. 감사합니다.