Real-time Trend Commercial Analysis

실시간 빅데이터 처리를 통한 상권정보 분석 시스템

소프트웨어융합대학 컴퓨터공학과 2019110634 이창렬
소프트웨어융합대학 컴퓨터공학과 2019102241 허인경

요 약

최근 창업을 하려는 사람들이 늘어나고 업종도 다양해지고 있다. 매물에 대한 단편적인 정보들은 존재하지만 매물과 상품 정보가 통합되어 시각화 된 서비스는 찾기 힘들다. 따라서 본 주제는 매출과 유동인구, 상권분석 빅데이터를 spark와 kafka를 이용해 정확한 실시간 결과를 안정적으로 도출하고 현 위치의 매물에 추천 업종을 제안해주는 서비스를 다루고 있다.

RTC는 예비 창업자에게 필수적인 위치 정보를 종합적으로 보여줄 수 있는 소프트웨어로 상권 정보, 접근성, 인기도를 토대로 사용자에게 어떤 업종이 사용자가 검색한 위치에서 최적의 업종인지를 알려준다. 본 논문에서는 즉각적으로 많은 데이터를 처리하며 최단의 시간에 원하는 정보를 얻을 수 있는 소프트웨어를 설계한다.

1. 서론

1.1. 연구배경

창업이 활발해지는 요즘 예비 창업자들은 많은 정보가 없는 상태에서 아이디어만을 가지고 시작을 한다. 아이디어는 본인의 분야에서 비롯된 것이라 사업적으로 필요한 정보들은 부족한 상태라고 가정하고 사업에 가장 기본적인 "위치"에 대해서 정보를 제공하고자 이 앱을 개발하게 되었다. 위치에는 다양한 정보들이 포함되어 있다. 예를 들어 어떤 상권에 위치하는지, 어떤 연령대의 사람들이 사는지, 어떤 집들이 주위에 있는지 등이 있다. 이러한 정보들은 개인이 일일이 찾는다면 얼마든지 찾을 수 있지만, 1 분 1 초가 중요한 사업에서 예비 창업자들은 시간 낭비를 원하지 않고 기술적으로 매분 매초의 흐름을 따라가기 어렵다. 따라서 우리는 창업자들이 사업 시작 전에 필요로 하는 정보를 우리 서비스를 통해 찾을 수 있게 했다. 기업의 존재는

한정적이고 그에 따라 사람들은 큰돈을 벌기 위해서는 취업보단 창업을 하는 사람들이 많아지고 있다.

중보벤처기업부에서 발표한 창업기업 동향 자료를 따르면 유독 20 대의 창업 증가율이 증가했음을 확인할 수 있는데 이는 취업 준비생이 선망하는 기업의 존재가 한정적이기에 취업을 포기하고 창업에 도전하는 사람들이 많아지고 있음을 확인할 수 있는 대목이다. 취업을 포기하고 창업으로 노선을 바꾼 20 대 창업가들은 40-60 대 창업가들에 비해 창업 노하우가 부족할 확률이 높고 자신이 창업하고자 하는 업종에 최적화된 오프라인 매장을 선택하기 어려워할 수 있다. 도움을 받고자 관련 서비스를 찾게 되지만 부동산 상가 매물 정보를 제공하는 서비스는 한정적이고 이러한 서비스는 매물 정보 제공에만 초점을 맞춰 여러가지 데이터를 연결하고 매칭하기 어려운 상황에 직면하게 된다. 따라서 기존의 부족한 서비스 때문에 부족한 초기 자금 리스크를 안고 여러 곳에 포진해있는 유동 인구나 근처 업종 정보, 상권 정보와 같은 객관적인 수치들을 계산하여 상가 매물을 선택해야 하는 창업가들을 위해 매물 정보를 포함해 창업에 필요한 데이터를 통합적으로 제공하고자 한다.

우리가 해당 프로젝트를 기획하는 단계에서 크게 영향을 받았던 것은 User Acquisition 마케팅에서 사용하는 데이터 레이크이다. 수많은 기기에서 앱을 열고, 장바구니에 담고, 물건을 사는 행위 등이 로깅 데이터로 들어오는 등 다양한 정보들이 전세계에서 실시간으로 날아온다. 하지만 다양한 곳에서 서로 다른 데이터를 가져올수록 우리가 원하는 데이터를 한곳에 모으는 과정에서 유실될 가능성이 높다.

우리는 이러한 데이터를 Kafka 를 통해 유실되지 않게 데이터베이스로 넘기고 싶었다. 이 과정 속에서 많은 양의 데이터를 실시간으로 처리하기 위해서 EC2 의 배치를 사용하기보다 Spark 를 통해 in-memory 방식의 장점을 활용하여 최대한 많은 데이터를 처리하고 싶었다. 실제 UA 마케팅에서는 전날의 사용자 데이터 리포트가 나와야 하기 때문에 수많은 데이터를 다음날까지 처리해야 한다. 이렇게 되면 컴퓨팅 리소스를 많이 사용하여 계산을 하는 것보다 spark 같은 소프트웨어에게 넘겨서 빠르게 처리하는 것이 비용적으로도 유리하기 때문이다.

1.2. 연구목표

첫째, 기존의 업종을 그대로 이어가는 것을 추천하지 않고 실질적인 현재 실제 주위 상권을 분석하여 적합한 업종을 추천해줄 수 있도록 하여 창업자들의 매장 선택에 도움이 되고자 한다.

둘째, 창업자들을 대상으로 단순히 매물 가격뿐만 아니라 근처 상권의 특징을 분석하고 관련 업종에 대해 정보를 제공하는 것을 목표로 한다. 이를 위해 지역 정보, 유동 인구, 매물 정보와

같이 실시간으로 들어오는 데이터를 Spark 와 Kafka 를 이용해 실시간으로 처리하여 사용자에게 최적화된 데이터를 제공하고자 한다.

셋째, Youtube 나 tweeter 등 각종 SNS 와 영상 플랫폼에서 지역 내의 유사 업종에 대한 사람들의 실시간 반응을 통해 목표 시장을 정의하고 목표 고객을 정의하는데 도움이 되고자 한다.

넷째, 실질적인 구현에 있어서 우리가 얻을 수 있는 ua 데이터가 없어 우리는 이러한 모델을 가정하며 서울시 데이터에서 최대한 많은 양의 데이터를 가져오는 것을 목표로 하였다.

다섯째, 앞서 가져온 많은 양의 데이터를 최대한 유실없이 빠르게 저장소에 가져오는 것을 목표로 하여 전송하는 데 있어서 시간과 최대 최소 전송크기에 제한을 두었다.

마지막으로 CLI 나 Django 등을 통해 우리가 계산한 데이터를 시각화 하여 사용자가 원하는 데이터를 조회하면 그에 맞는 관련 데이터를 보여줄 수 있도록 구현하는 것을 최종 목표로 삼았다.

1.3. 연구목표

위의 목표를 구현하기 위한 기술적인 흐름은 다음과 같다.

2. 관련연구

2.1. Main Frame

2.1.1 데이터 처리

우리는 기업에서 보유중인 데이터처럼 많은 데이터를 가지고 있지 않기 때문에 가능한 한 다양한 곳에서 많은 양의 데이터가 넘어온다는 상황을 가정하고 싶었다. 따라서 데이터를 최대한 잘게 쪼개서 많은 양의 데이터가 비슷한 시간에 한 번에 넘어온다고 가정했다.

2.1.2 Kafka

Kafka 는 데이터를 만들어내는 Producer 와 데이터를 소비하는 Consumer, 그리고 이 둘 사이를 중재하는 Broker 로 이루어진 Publish-Subscribe 모델을 구현한 분산 메시징 시스템이다. Kafka 는 클러스터 위에서 Producer 가 전송한 메시지를 Broker 가 위치한 서버의 파일 시스템에 저장해 한 Broker 에게 장애가 생기더라도 복사본을 Consumer 에게 전달할 수 있어 멱등성을 가지면서도 장애 상황에 유연하게 대처할 수 있다. 또한 scale in/out 에 따른 Consumer 의 처리 형태와 속도를 고려하지 않아도 되어 분산 및 복제 구성이 쉽다.

다양한 회사에서 각자 내놓은 kafka 의 차이점을 먼저 알고, 그에 추가적으로 사용할 수 있는 기능들을 파악했다. 여기서 말하는 회사들은 confluent, apache 등이 있는데 우선 각 회사별로 추가적으로 사용하는 기능들의 종류도 비슷하지만 사용법이 상이하기 때문에 되도록이면 같은 회사의 제품끼리 사용하는 것을 권장한다. Kafka 는 하나 이상의 topic 을 배치하여 producing 하며 그 topic 에서 들어오는 데이터 스트림을 어떻게 consume 할지 broker 과 zookeeper 를 통해 관리할 수 있다. Broker 는 topic 을 기준으로 메시지를 관리하며 producer 는 특정 topic 에 메시지를 생성하여 broker 에게 전달한다. Broker 가 전달받은 메시지들을 topic 별로 분류해 partition 들에 쌓아놓으면 해당 topic 을 구독하는 consumer 들이 메시지를 가져가서 처리하게 된다.

우리는 각 사이트들을 통해 가져온 데이터들을 미리 설정해둔 여러 개의 broker 와 zookeeper 를 이용하여 토픽에 produce 한다. 이때 어떤 데이터들이 현재 어떤 토픽의 어떤 파티션에 어떤 브로커를 통해서 들어왔는지 akhq 를 통해 모니터링을 할 수 있다. 그 후 consumer 에서 데이터를 알맞는 형태로 storage 에 넘겨준다. 여러 차례 kafka 를 실행시켰을때 몇몇 데이터는 용량이 생각보다 커서 topic, broker, producer, consumer 등의 properties 를 조정해주었고 결과적으로 원하는 만큼의 데이터를 유실없이 전달했다.

2.1.3 Spark

Spark 란 SQL, 스트리밍, 머신러닝 및 그래프 처리를 위한 빅데이터 분산 처리 엔진이다. In-memory 연산이 가능한 Spark 는 디스크 기반의 Hadoop 의 맵리듀스보다 100 배 빠른 속도로 워크로드를 실행할 수 있으며 정형 또는 비정형 데이터로부터 가치를 고속으로 처리하여 데이터 실시간 스트리밍 처리를 가능하게 한다. Spark 는 데이터 분석 작업에 필요한 통합 API 를 제공하는 통합 엔진 기반 자체의 라이브러리이기 때문에 다양한 기능과 접목할 수 있으며 이러한 기능을 하드웨어의 제한을 벗어나 자유롭게 사용할 수 있다.

Spark 에서는 데이터를 처리하는 데 있어서 최선의 방법은 가장 최신의, 대량의 리소스를 입력하는 것이다. 하지만 그렇게 된다면 비용적으로 부담이 되기 때문에 데이터 양을 확인하고 어느 순간에 비용 극소점을 찾는 것이 중요한데 spark 를 구현할 때 이 부분을 가장 많이 신경 썼다.

2.2. Sub Frame

2.2.1. Django

Django 는 파이썬 무료 오픈소스 웹 어플리케이션 프레임워크로 다양한 웹 서버의 요청을 처리해주는 프레임워크이다. 웹 개발에 필요한 많은 기능이 구현되어 있어 다양한 종류의 웹 사이트를 빌드하는데 사용할 수 있고 확장성 덕분에 트래픽에 대응하여 크기를 조절할 수 있다. 또한 Python 으로 작성되어 많은 플랫폼 사이에서 동작할 수 있을 뿐더러 유지보수가 쉽고 재사용하기 좋기 때문에 장고를 통해 웹 개발을 손쉽게 진행할 수 있다.

2.2.2. graphDB

그래프 데이터 베이스는 그래프 이론에 토대를 둔 NoSQL 로 관계를 저장하고 탐색하도록 구축되어 있는 데이터 베이스이다. 데이터 간의 관계에 초점을 둔 DB 로 node 를 사용하여 데이터 entity 를 저장하고 edge 로는 entity 간의 관계를 저장하여 직관적으로 모델링할 수 있다. 또한 NoSQL 이기에 스키마가 없는 구조로 데이터 용량이 늘어나거나 입력되는 형태가 다양해져도 수용 가능하며 질의 처리 속도가 빠르다는 장점이 있다.

2.3. 기존에 제공하고 있는 서비스에 대한 연구

기존 소상공인시장진흥공단에서 제공하고 있는 데이터는 통계 자료를 기반으로 한 상권 현황에, 사기업 서비스가 제공하고 있는 데이터는 중개업 서비스에 초점을 맞춰 실제 사용자가 지도를 기반으로 다양한 정보를 접근 가능하게 하고 있다. 우리는 이런 양면의 이점을 살려서 사기업 서비스를 모티브 삼아 소상공인에서 제공하는 통계 자료를 기존의 사기업 서비스에 접목시켜 단순히 매물의 정보뿐만 아니라 트렌드까지 제공한다.

3. 프로젝트 내용

3.1. 시나리오

3.1.1. Search Place

[그림 1] 프로젝트 사이트에 접속하면 보이게 되는 메인 화면

[그림 1]는 프로젝트의 메인 화면이다. 사용자가 위치를 검색하면 지도에 해당 위치와 함께 근처 상권, 유동 인구, 매물 정보, 근처 인기 상권 순으로 나온다.

- 근처 상권으로는 개수와 매출액 기준으로 상위 3 개의 리스트와 그래프가 나오며, 리스트의 업종을 누르면 3.1.2 의 근처상권 조회로 넘어간다.
- 유동 인구는 근처 역, 정류장, 주거인구를 토대로 구역별로 인구수를 보여준다.
- 매물 정보는 해당 매물의 매매금, 권리금, 고정 지출 비용인 관리비가 나오며 추가적으로 계약 면적이 나온다.
- 근처 인기 상권은 실시간으로 최근 트렌드가 어떤 지 각종 SNS 의 조회수를 보여준다.

3.1.2. 근처상권 조회

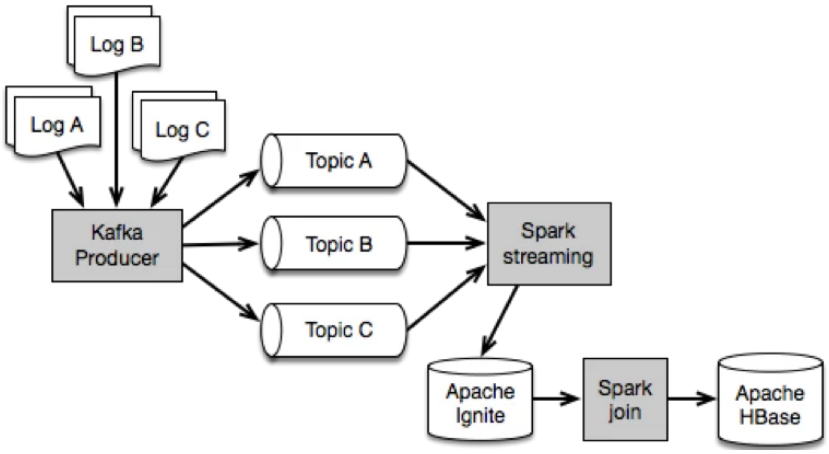
[그림 2] 근처 상권의 리스트를 클릭했을 때의 화면

근처 상권의 리스트를 누르면 해당 업종에 대해서 위치 및 매출정보가 나온다. 이를 통해 사용자가 선택한 업종의 평균매출을 알수 있고, 3.1.1 에 나온 고정 지출비용과 비교해서 사용자가 실제로 어떤 업종이 더 효율적인지 선택할 수 있게끔 한다.

3.2. 요구사항

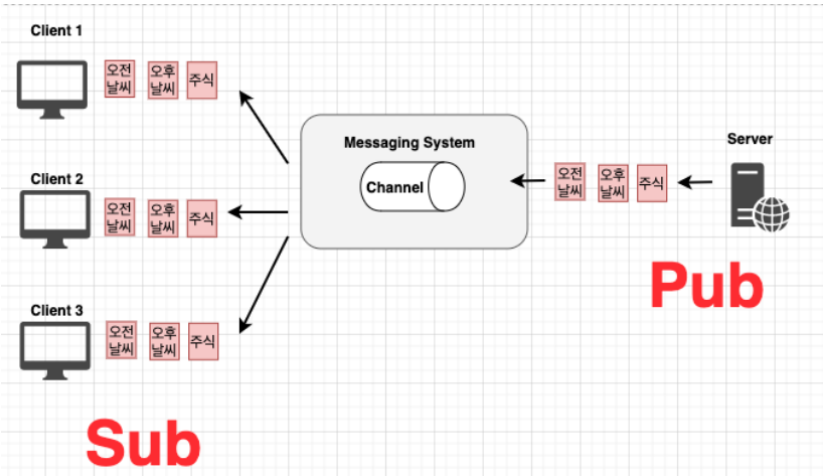
3.2.1. Kafka, Spark 에 대한 요구사항

- 쓰기가 아닌 읽기에 초점을 맞춰 메세지를 Broker 에서 Consumer 로 대응하는데 쓰이는 Latency 를 최적화한다.
- 모든 데이터는 중간 과정에서 잃어버리지 않아야 한다. 이를 위한 보증 과정 속에서 엄격한 트랜잭션 관리로 인한 오버 스펙보다 높은 처리를 우선순위로 해야 한다.
- 프론트엔드에서 요구하는 반응 속도를 맞추기 위해 세션 타임 아웃 시간을 적절하게 설정해 장애상황을 최소화하고 리밸런싱을 시도하여 available 한 데이터를 꾸준히 제공할 수 있도록 한다. 동시에 네트워크를 통해 받아오는 데이터를 실시간 스트림 처리할 수 있도록 한다.
- 데이터가 유독 많거나 적은 지역을 탐색하게 된다면 Kafka 가 자동으로 스케일 조정을 하여 실시간으로 대응할 수 있도록 한다.
- 여러 Consumer 들이 처리한 데이터를 병목 현상을 겪지 않으면서 Spark 와 연동하여 확장성을 보장한다.



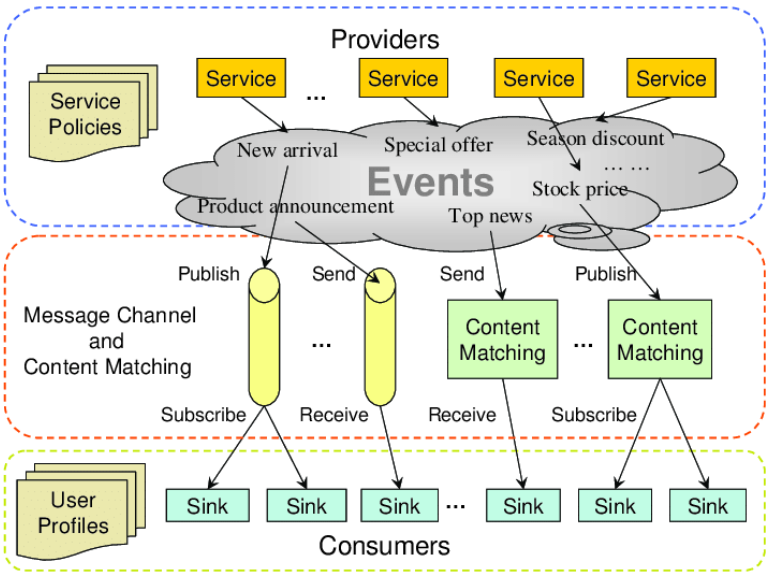
[그림 4-1] Kafka 와 Spark 의 연동 요구사항

- Publisher Subscriber 모델 구조를 통해 Publisher 나 Subscriber 가 죽더라도 안정적으로 데이터를 처리할 수 있어야 한다.



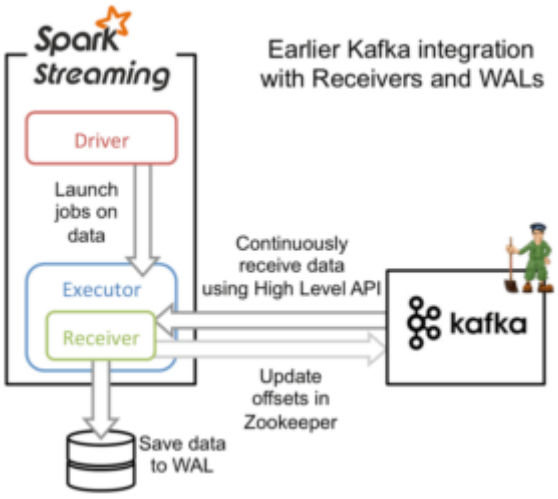
[그림 4-2] Kafka 와 Spark 의 연동 요구사항

- Filtering, 정렬, 집계, 조인, 정리, 중복 제거 연산을 하는 과정 속에서 데이터를 수정하고 데이터를 새 데이터 저장소로 이동하는 ETL 과정에서 Kafka 에게 받은 데이터를 연산하기 쉽게 전처리해야 한다.
- Event-driven 구조로 구현하여 빅데이터 연산으로 인한 교착상태에 빠지지 않도록 설정해야 한다.



[그림 4-3] Kafka 와 Spark 의 연동 요구사항

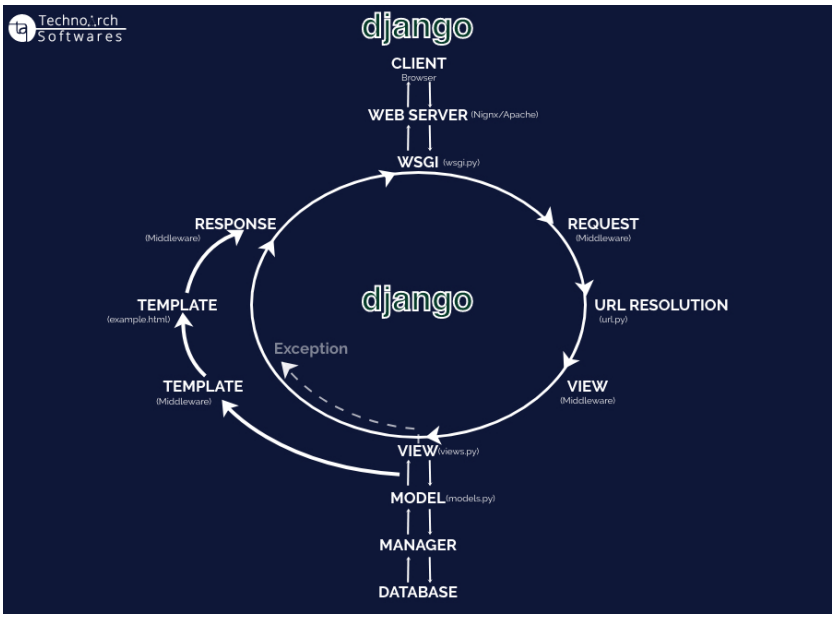
- 일괄 처리를 통해 프론트엔드에서 요구하는 반응 속도를 맞춰야 한다.
- 복원성을 보장하여 연산 과정에서 오류가 나더라도 빠르게 장애가 나기 전 계산 완료한 데이터에 접근할 수 있어야 한다.
- 워크 플로우 오케스트레이션을 통해 자동으로 자원 할당을 조절하여 개발자는 개발 외의 시간을 아낄 수 있도록 한다.
- 일정량 또는 일정기간 동안 데이터를 모아서 한꺼번에 처리하는 일괄처리(Batch Processing)와 달리 연속되는 실시간 데이터를 빠르고 효율적으로 처리할 수 있어야 한다. 스트리밍 데이터를 활용하는 것이므로 데이터 공유에 이점이 있기 위해서는 특히 속도가 보장되어야 한다.
- Zookeeper 를 통해 시스템 간의 정보 공유, 상태 체크, 서버들 간의 동기화를 위한 락 등을 처리해주는 서비스를 쓰지만 요새는 Zookeeper 가 점점 없어지는 추세이므로 프로젝트 중 개선해 나갈 예정이다.



[그림 4-4] Kafka 와 Spark 의 연동 요구사항

3.2.2. Django 에 대한 요구사항

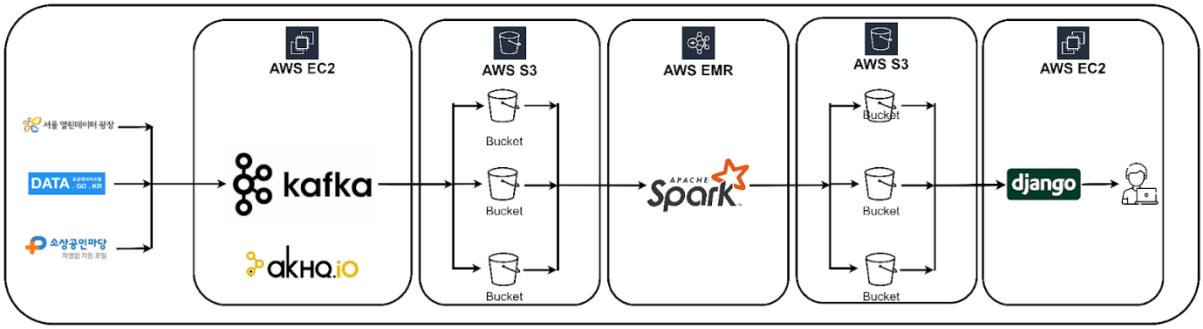
- 사용자가 요청한 지역에 대해 필요한 정보를 빠르게 취합하여 요청을 보내고, 받은 응답을 지연없이 사용자에게 보여준다.



[그림 5] Django 요구사항

3.3 시스템 설계

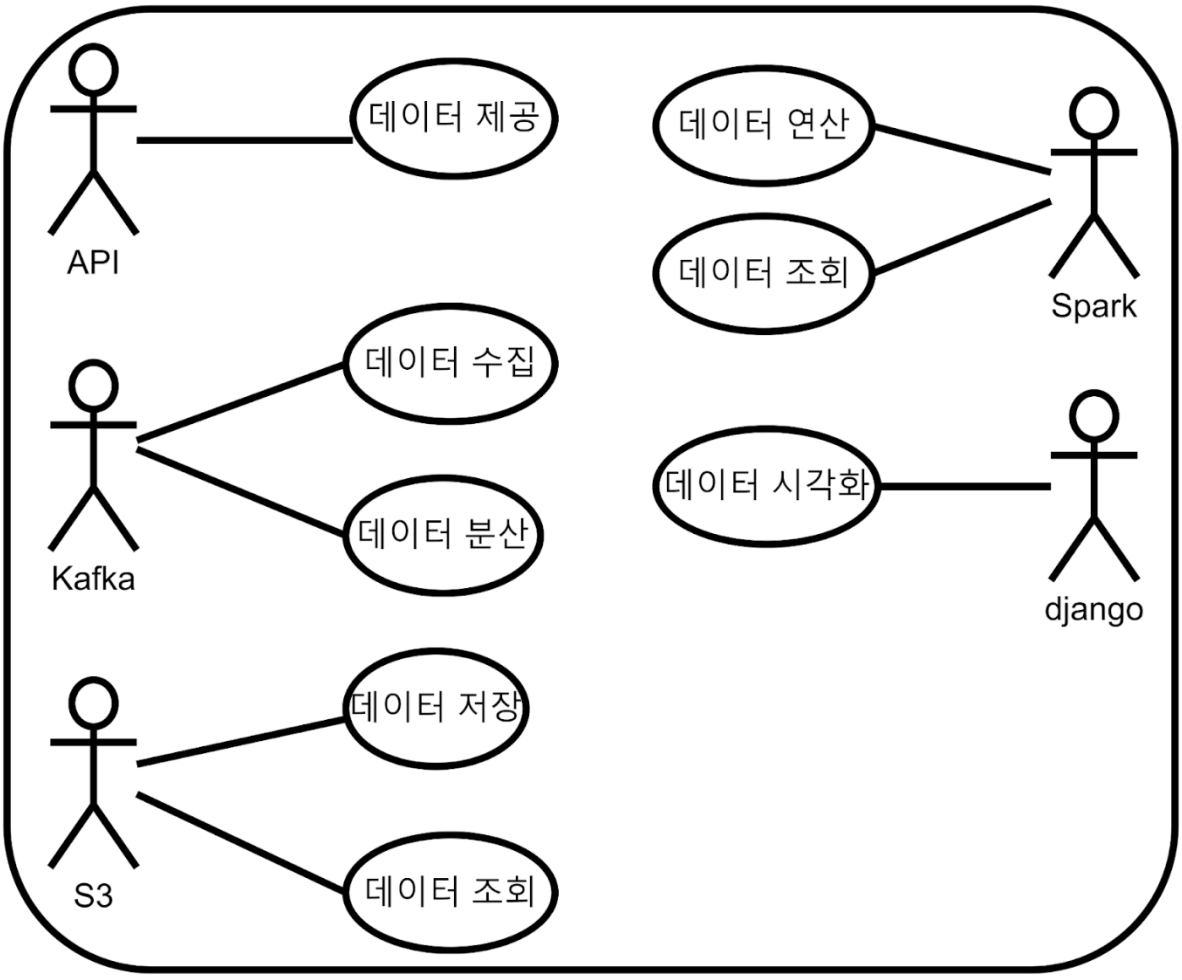
3.3.1 시스템 구성도



[그림 6-1] 시스템 구성도

시스템 구성도는 그림[6-1]과 같다.

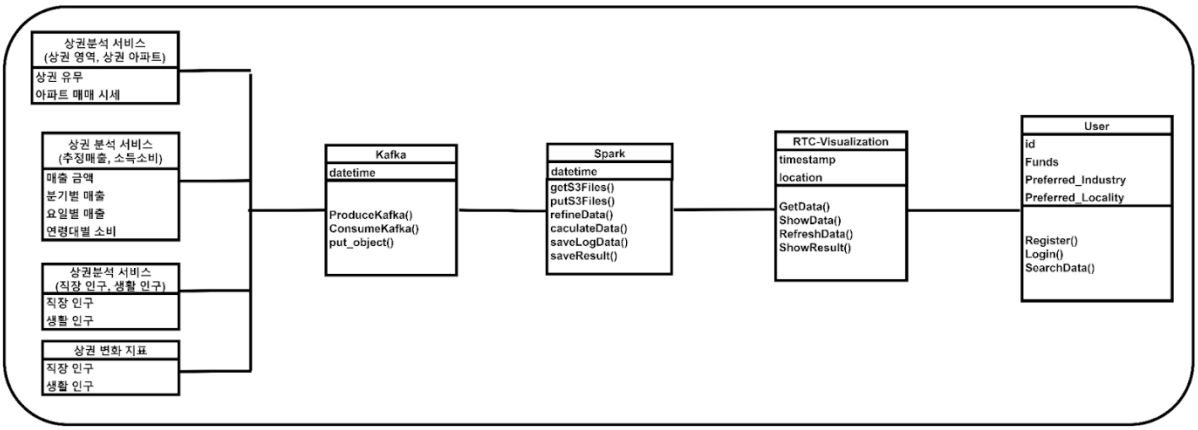
3.3.2 Use Case Diagram



[그림 6-2] Use Case Diagram

Use Case 는 5 개의 액터로 이루어져있다. 먼저 많은 사이트들의 데이터를 API 를 통해 우리 시스템으로 가져온다. 이를 Kafka 가 전달받아 데이터를 수집하여 분산처리를 해준다. Kafka 를 통해 분산된 데이터는 S3 가 저장하여 우리 서비스의 모든 데이터는 S3 에 모이게 된다. Spark 는 이 S3 에 있는 데이터를 조회하여 서비스 목적에 맞게끔 정제 및 연산을 한다. 마지막으로 Django 는 S3 로부터 데이터를 가져와 데이터 시각화를 한다.

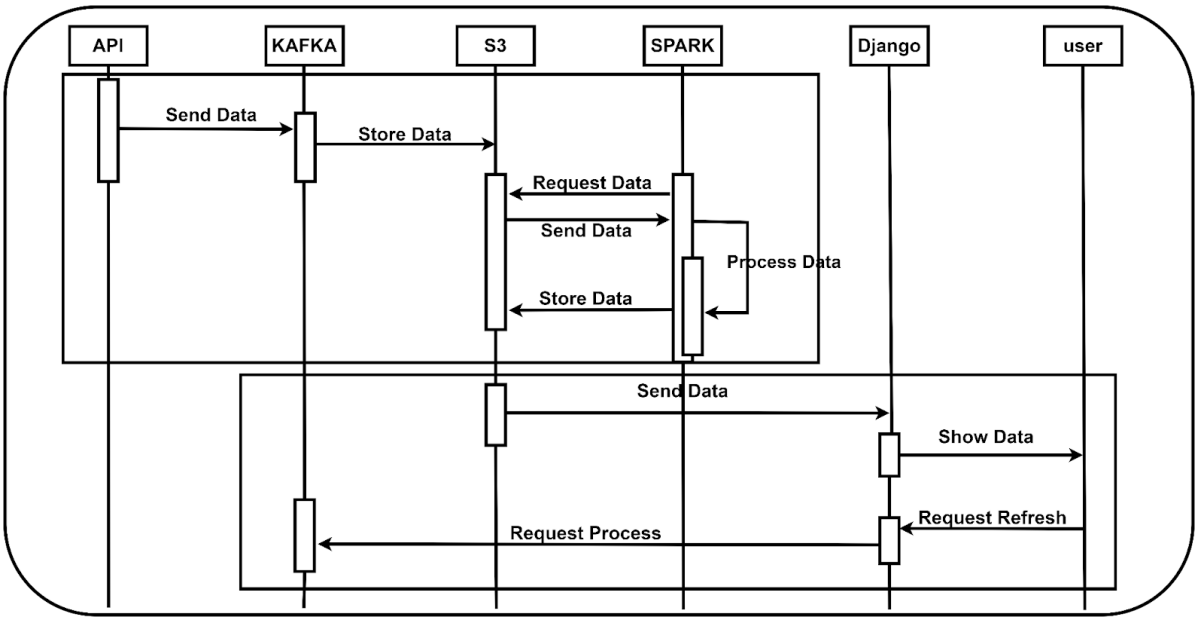
3.3.4 Class Diagram



[그림 6-3] Class Diagram

클래스는 크게 API, Kafka, Spark, Django, User 로 이루어져 있다. 각 사이트별로 API 가 핵심 키워드를 통해 데이터를 가져오면 Kafka 는 이를 안전하게 전달해준다. Spark 는 전달받은 데이터를 가지고 연산하고 Django 는 사용자가 검색하는 정보를 검색시간과 함께 제공한다. 사용자는 초기에 입력한 본인의 자금, ID 를 가지고 서비스를 사용할 수 있다.

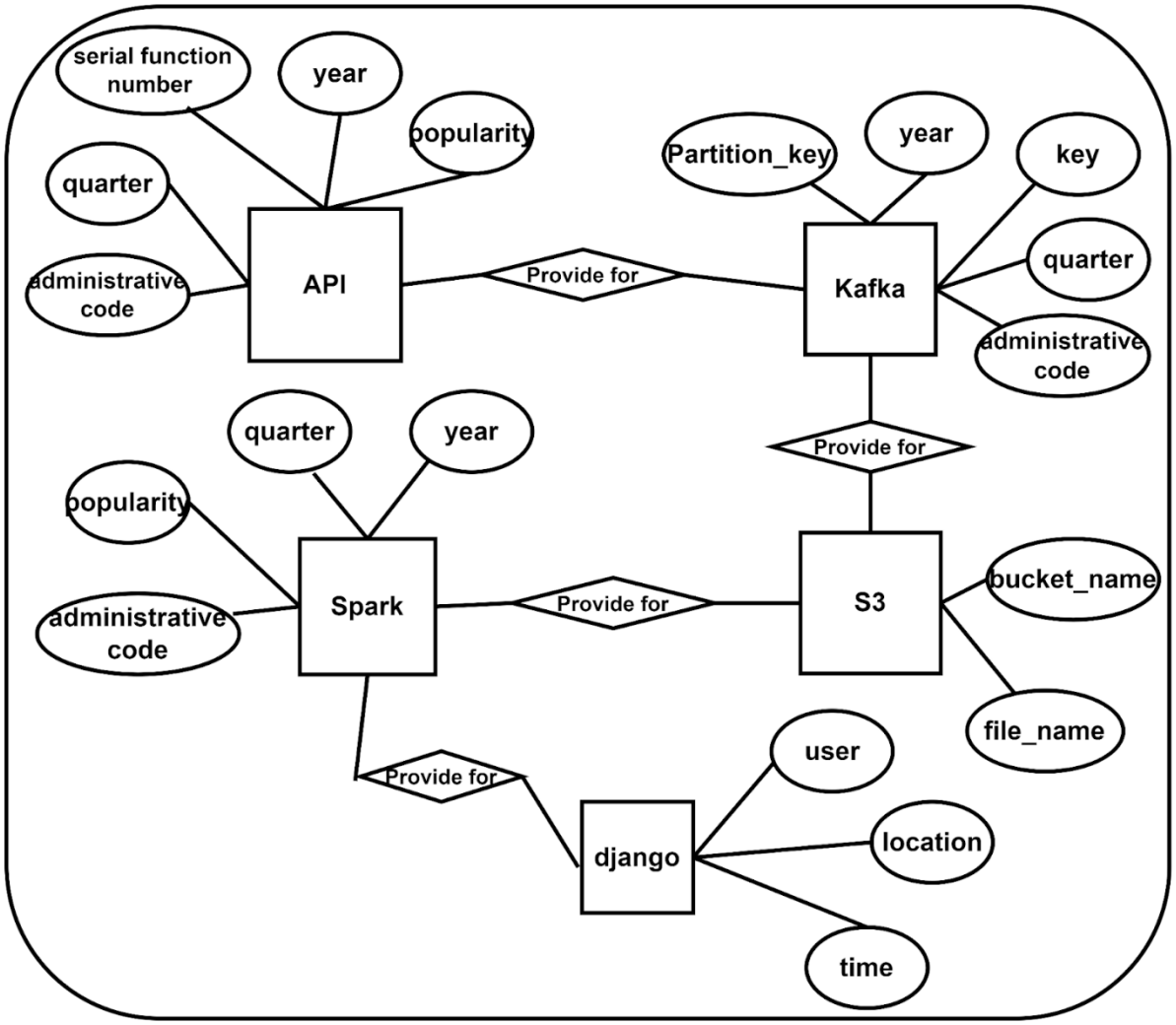
3.3.5 Sequence Diagram



[그림 6-4] Sequence Diagram

본 서비스의 전반적인 흐름을 보여주는 Sequence Diagram 이다. API 는 Kafka 에게 데이터를 전송한다. 전달된 데이터는 S3 에 저장이 되고, Sparks 는 구동될 때 S3 에게 데이터를 요청한다. Spark 가 데이터를 S3 로부터 전달받으면 자체적으로 연산을 해 다시 S3 에게 데이터를 넘겨준다. S3 에 정제된 데이터를 Django 에서 가져가고 이를 사용자에게 보여준다. 사용자는 Django 에게 갱신명령을 내릴 수 있고, 이를 통해 위의 과정이 즉시 반복되어 사용자가 보는 화면이 갱신된다.

3.3.6 ER Diagram



[그림 6-5] ER Diagram

다음은 ER Diagram 이다. 모든 개체는 년도, 분기, 행정동 코드를 기본키로 가지고 데이터들을 전달한다. 먼저 API 는 기본키와 각 사이트별로 할당된 키들을 가지고 데이터를 Kafka 에게 전송한다. Kafka 는 API 로부터 전달받은 데이터에 Partition key 를 추가해 데이터를 전달한다. 이때 기본키는 JSON value 에서의 키 값이 된다. S3 는 Kafka 로부터 전달된 데이터를 저장하게 되는데, 파일명은 데이터별로 가져온 사이트의 키 값을 참고해 생성했고, 버킷 명은 Kafka 의 토픽명을 참고해 생성했다. Spark 는 S3 로부터 데이터를 요청하여 자체적으로 정제하고 이 정제된 데이터를 Django 에서 사용자가 본인이 필요한 위치에 맞게 검색하여 결과적으로 사용자의 검색시간, 본 서비스에서의 결과값 등이 웹 상에 보여진다.

3.4. 구현

소스코드 경로 및 폴더 구성은 아래와 같다.

경로 : https://github.com/capstonedesignRTC/RealtimeTrendCommercial

- Kafka 폴더

파일 위치	설명
/utils/seoul_api.py	서울시 우리마을가게 상권분석서비스 API를 사용하기 위한 모듈이다.
/utils/data_go.py	공공데이터 포털 서비스 API를 사용하기 위한 모듈이다.
producer.py	kafka/utils/ 폴더에 위치한 API 모듈 파일을 이용해 Procuder에게 API 데이터를 전송하는 모듈이다. 어느 데이터를 가져올 것인 것인지, 어느 기간의 데이터를 Producer를 통해 보낼 것인지 설정되어 있다.
consumer.py	Kafka broker가 처리한 데이터를 받아와 S3에 저장하는 Consumer가 선언된 모듈이다.
docker-compose.yml	Apache Kafka를 전신으로 두어 Apache Kafka 내의 config 폴더의 properties 파일들을 알맞게 설정한다. 이 파일에는 Kafka 매니져 프로그램인 Apache Kafka HQ(AKHQ)에 관련된 설정도 포함되어 있어 클러스터를 띄울 때 따로 AKHQ를 실행시키지 않아도 된다.

그 외에도 환경 설정을 위한 파일과 Kafka 실행 시 어떤 명령어를 쳐야 Producer와 Consumer가 실행되는지 정리한 명령어 파일이 있다.

- spark 폴더

파일 위치	설명
/spark/configure.py	Spark를 실행할 때 효율적인 Spark Job을 만들기 위해 설정한 properties 값들이 있다. 현재 설정된 값은 프로젝트 실행을 위한 최적화된 값이다. 또한 Hadoop API를 이용해서 S3에 있는 데이터를 가져오는 함수와 결과값으로 도출된 Data를 S3에 저장하는 함수에 관한 정의도 포함되어 있다.
/modules/dataframe.py	S3에 저장된 데이터를 정제하는 모듈이다. 필요한 열을 분리하고 열 이름을 변환하여 spark SQL 연산 시 데이터 사용을 용이하게 해준다.

/modules/calculation.py	데이터를 S3에서 가져와 정제시키고 데이터 연산 함수를 호출하여 분석을 진행한 뒤, 결과 값을 저장하는 과정이 실행되는 모듈이다.
main.py	Spark를 실행시키기 위한 루트 파일이다. 이 파일을 실행함으로써 사람의 개입없이 연속적인 데이터 처리가 시작된다.
Dockerfile	Spark를 실행시키기 위해 설치되어야 하는 Java, Hadoop 및 Jar들의 dependency가 정리되어 있어 실행 시 추가적인 설치 없이 spark를 실행시킬 수 있다.
emr_start.sh	EMR 사용할 때, CI를 위한 쉘 스크립트이다. 쉘 스크립트를 EMR 인스턴스의 인자로 넘겨주어 자동으로 실행되게 한다면 사람의 개입없이 EMR에서 실행 환경을 준비하고 main.py을 시작한다.

그 외에도 Spark 환경설정을 위한 파일과 AWS S3에 접근하기 위해 정의된 credential 파일이 정리되어 있다.

- Django 폴더

파일 위치	설명
/main/s3storage.py	Kafka가 저장한 raw-data와 spark가 도출해낸 결과 데이터에 접근할 수 있는 모듈 파일이다.
/main/view.py	사용자가 입력한 값을 받고 연산한 결과를 전달해주는 모듈이 정리된 파일이다.

그 외에도 사용자가 Kafka-Spark 데이터 레이크에서 얻은 결과값을 시각적으로 확인할 수 있는 웹 프레임워크 관련된 파일이 있다.

4. 프로젝트 결과

4.1. 연구 결과

Kafka 의 Producer, Consumer, Server 에서 데이터 송수신 크기와 데이터 offset 에 관련된 properties 값을 설정하여 데이터가 빠르게 데이터 파이프라인에 들어갈 수 있게 하였고 설정된 properties 값에 따라 Broker & Zookeeper 의 개수를 맞춰 데이터가 유실되지 않도록 설정하였다.

동시에 저장된 데이터를 Spark 가 최소한의 하드웨어 리소스를 사용하면서 대용량 연산을 진행할 수 있도록 spark configuration 값을 설정하여 최적 spark job 을 실행할 수 있도록 하였다.

4.1.1 Kafka 에서의 Properties 설정

여러 가지 경우의 수를 두고 크기를 적절히 배합해가며 데이터를 전송했을 때, 온프레미스의 경우 메모리나 CPU 의 한계를 가질 수밖에 없다. 따라서 서버가 안정적으로 구동가능한 범위를 확인해야 된다. 본 서비스는 여러 config 들을 통해 서비스 환경에 맞는 최적의 설정을 찾았고, 이는 producer, server, consumer 에서 모두 최대 message 크기, buffer 크기를 32MB 로 두어 유실없이 안정적으로 데이터를 전송함에 성공했다.

또한 메시지를 잘게 자르는 방법도 도입했는데, 비록 시간적인 효율은 더 좋은 성능의 로컬에서 돌리는 것 보다는 떨어지지만, 안정성을 최종 목표로 삼고 있는 서비스라 이를 최종 설정으로 두었다.

따라서 수정한 properties 값 중, 성능에 크게 영향을 미쳤던 값들을 정리하면 다음과 같다.

어플리케이션 명칭	명칭	설정 값
producer	max.request.size	33554432 (bytes)
producer	batch.size	33554432 (bytes)
producer	buffer.memory	33554432 (bytes)
server	socket.send.buffer.bytes	33554432 (bytes)
server	socket.receive.buffer.bytes	33554432 (bytes)
server	message.max.bytes	20971520 (bytes)
consumer	fetch.max.bytes	52428800 (bytes)
consumer	max.partition.fetch.bytes	1048576 (bytes)
consumer	fetch.min.bytes	16384 (bytes)
consumer	enable.auto.commit	false
consumer	auto.offset.reset	earliest

Default 값인 1048588 bytes 을 32 MB 로 수정하여 안정적으로 데이터를 전송할 수 있도록 설정하였고 consumer 가 데이터를 받지 못한 상태에서 다시 producer 를 시작하였을 때 중복되지 않도록 offset 의 자동 commit 을 꺼주는 enable auto commit 값을 false 로 설정하였다.

4.1.2 프로젝트 최적 Spark Configuration 설정

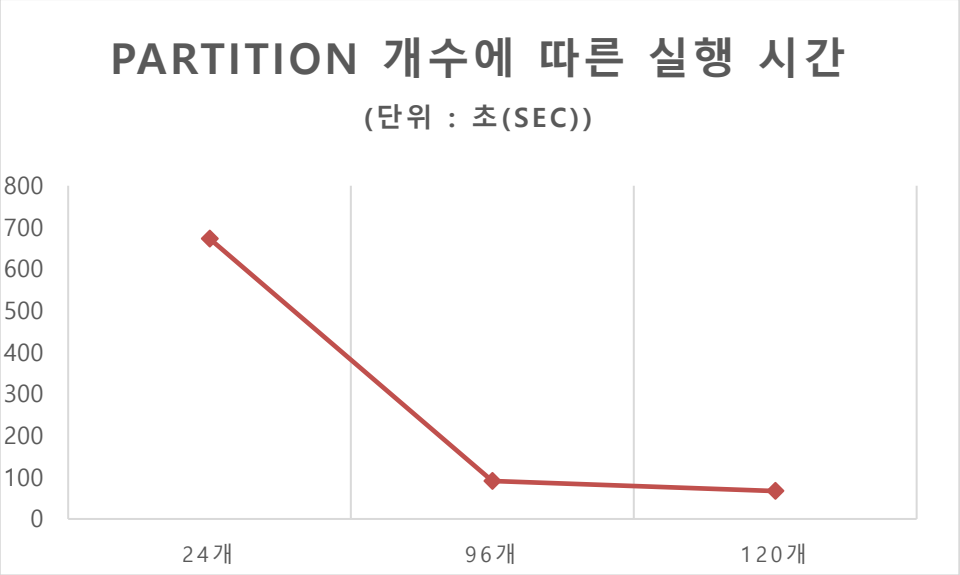
Spark 는 하둡의 맵리듀스 작업에서 성능의 병목현상으로 지목되던 디스크 I/O 비용을 최소화하고 데이터 분석 작업에 용이한 인메모리 컴퓨팅 기반의 범용적 데이터 분산처리 시스템이다. Default 설정으로 Spark 연산을 진행하게 된다면 이러한 spark 의 이점을 살리지 못하게 된다. 따라서 주어진 하드웨어 리소스와 데이터에 맞게 configuration 을 설정하여 Spark cluster 를 실행시키기 위해 연구를 진행하였고 크게 2 가지를 수정함으로써 성능 향상을 이끌었다.

첫번째로 적절한 메모리 값 설정을 통해 성능 최적화를 이끌어낼 수 있었다. Storage memory 값을 크게 줄 수록 cache 된 데이터를 많이 쌓을 수 있어 broadcast 변수를 많이 저장할 수 있게 되었고 이는 I/O 연산을 줄이는 결과를 가져왔다. 또한 Execution memory 값을 크게 줄수록 map 수행으로 인한 결과들이 shuffle intermediate buffer 에 많이 저장되어 디스크로의 spilling 을 줄일 수 있게 되었다. 결과적으로 default 로 선언된 메모리 값으로 동작시킬 때에 계속 발생하던 메모리 부족으로 인한 스파크 작업 실패를 방지할 수 있었다. 하지만 하나의 Executor 에 memory 값을 크게 주었더니 garbage collection 딜레이가 발생하여 성능이 저하되는 것을 발견하고 계속 값을 바꿔가며 연산을 진행해본 결과, 프로젝트에서는 spark.driver.memory 와 spark.executor.memory 를 각각 14GB 씩 부여하는 것이 최적임을 알 수 있었다.

두번째로 partition 개수를 수정하여 성능을 끌어올렸다. 프로젝트 데이터를 처리할 때 join, union, groupBy, sort, coalesce, repartition 연산이 있었다. 이러한 연산들은 물리적인 데이터의 이동이 있어야 하기 때문에 spark 에서 데이터를 재분배하는 과정을 불러일으키게 된다. 자주 데이터를 재분배하게 되면 실제 연산 시간보다 부가적인 시간이 추가되기 때문에 partition 개수를 연산 함수 실행 전에 적절한 크기로 제어하는 것이 cluster 를 실행시키는데 있어 큰 영향을 주게 된다.

프로젝트 데이터 연산을 위해서는 data shuffle 을 유발하는 Wide 변환 함수가 자주 실행되어야 했으므로 이러한 연산 함수를 사용하지 않는 대신, data shuffle 을 통한 repartition, disk partitioning 을 진행하는 partitionBy 함수를 통해 데이터를 저장하기 전에 partitioning 을 진행하였고 예상보다 성능이 많이 향상되어 partition 개수 설정에 관한 심화된 실험을 진행하였다.

Spark.driver.memory, spark.executor.memory 의 값을 각각 4GB 두고 spark.executor.cores 를 4 로 설정한 뒤, spark.sql.shuffle.partitions 의 수를 각각 24, 48, 72, 96, 120 으로 두고 총 256MB 크기의 데이터를 연산 처리해보았다. 이 중 24, 96, 120 으로 값을 두었을 때를 그래프로 정리하였다.



[그림 7] Partition 개수에 따른 실행 시간 결과 그래프

Partition 개수를 증가시키면 증가시킬 수록 결과적으로 같은 양의 데이터라도 더 빠르게 처리할 수 있는 것을 확인할 수 있었고 memory limit over, memory spill 로 인한 자원 문제가 덜 발생하였다.

Partition 개수를 늘리는 것으로 각 executor 에서 한 번에 처리해야하는 양이 줄어들어 shuffle spill 이 덜 발생하게 되고 memory 부족 오류를 줄일 수 있었다. 하지만 partition 개수를 100 이상으로 설정하는 것이 성능 향상에 크게 도움이 되지는 않았다. 이는 모든 partition 을 scheduling 하기 위한 시간 증가와 작은 사이즈의 파일들을 생성하고 분배하면서 발생하는 I/O 연산으로 인한 시간 지연 때문에 기대한 만큼 성능 향상이 이뤄지지 않는다는 것을 실험을 통해 확인할 수 있었다.

그 외에도 성능 향상을 위해 추가적으로 값을 수정해보았는데 가장 크게 영향을 미친 것은 spark.executor.instances 값이었다. 프로젝트에서 사용하는 데이터는 크기가 작지만 많이 나뉘어 있어 이러한 데이터를 읽어오고 하나로 합치는데 I/O 작업이 많이 발생하였다. 따라서 executor 의 값을 키워 많은 양의 HDFS I/O 연산을 가능하게 하여 성능이 향상되었지만 과하게 값을 키울수록 같은 node 내의 executor 끼리의 통신 비용이 많이 들어 오히려 성능이 나빠지는

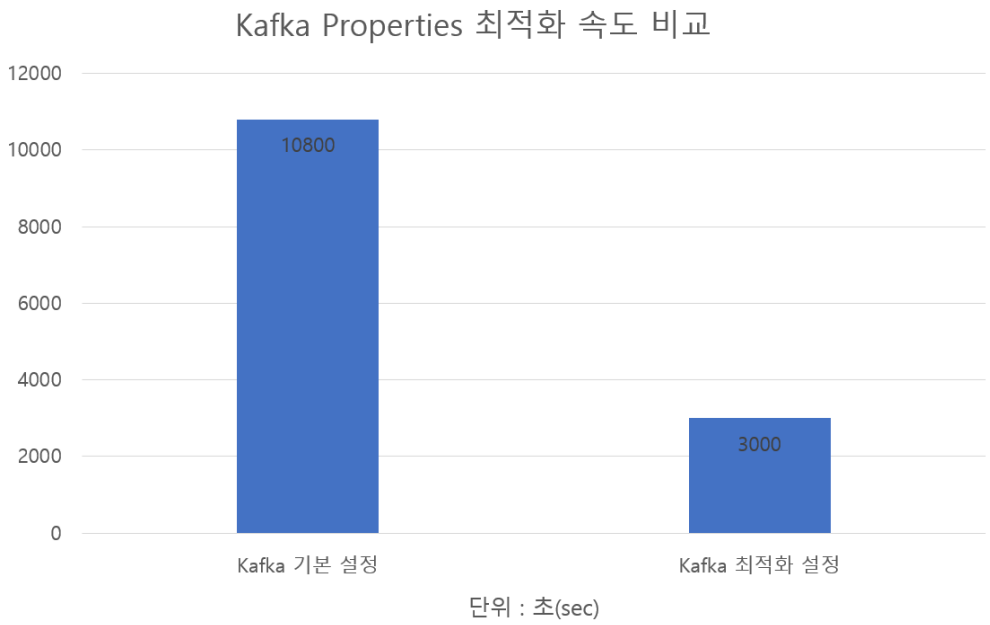
것을 확인할 수 있었다. 따라서 우리 프로젝트에서는 8 로 설정하여 최대한의 성능 향상을 가져왔다.

따라서 수정한 configuration 값 중, 성능에 크게 영향을 미쳤던 값들을 정리하면 다음과 같다.

명칭	값
spark.driver.memory	14 (GB)
spark.executor.memory	14 (GB)
spark.executors.cores	4
spark.executor.instances	8
spark.speculation :	false
spark.shuffle.compress	true

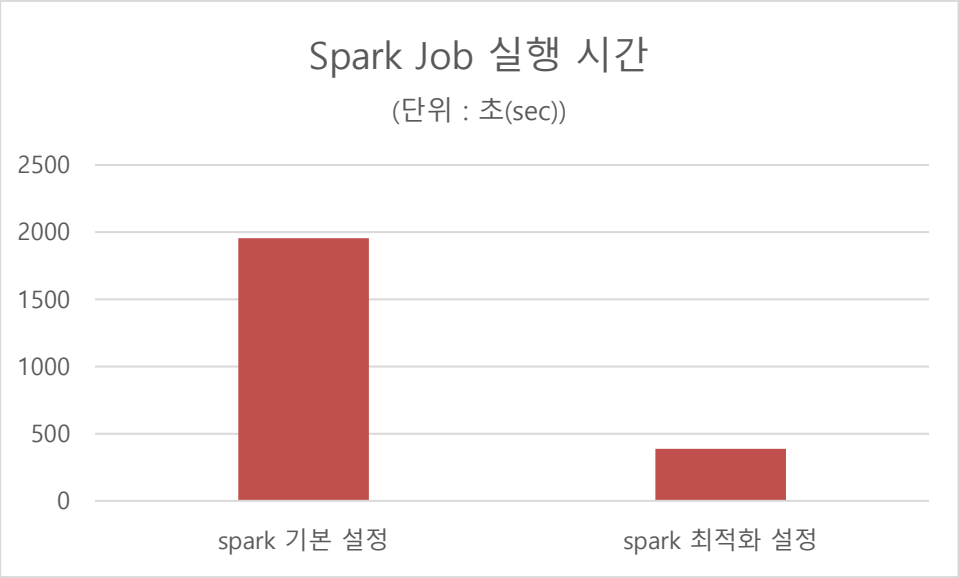
4.2 성능 평가

Kafka 의 경우 모든 API 를 한 세트 돌리는데 10800 초 가 SS 량 소요됐다. 즉, 본 서비스를 위한 데이터를 수집 시간이 최소 3 시간이라는 것이다. 하지만 각종 properties 들을 본 서비스의 환경에 맞게 설정해주고 각 데이터의 크기를 다양한 크기로 잘라서 전송해본 결과, 3000 초라는 시간이 나왔다. 단지 producer 에서의 전송만이 목적이라면 기존의 설정을 유지하는 편이 더 빠르다. 하지만 Kafka 는 다른 서비스를 필요로 하는 서비스로, 다른 서비스까지의 연결 시간을 전체 시간으로 생각하고 계산해본 결과 기존의 설정이 약 3 배 정도 오래 걸린다는 사실을 확인할 수 있었다.



[그림 8-1] Kafka Properties 최적화 속도 비교 결과 그래프

Spark 의 경우 총 256MB 크기의 데이터를 처리하는데 있어서 기본 설정 값을 사용하였을 때에는 평균적으로 2000 초 넘게 필요했다. 이 경우 spark 실행 도중에 memory out 으로 인해 작업이 완료되지 않고 spark 가 종료되어 정확한 실행 시간을 알 수 없었다. 반면 위에서 설정한 최적값으로 spark 실행시 평균적으로 약 400 초가 필요함을 확인할 수 있었다.

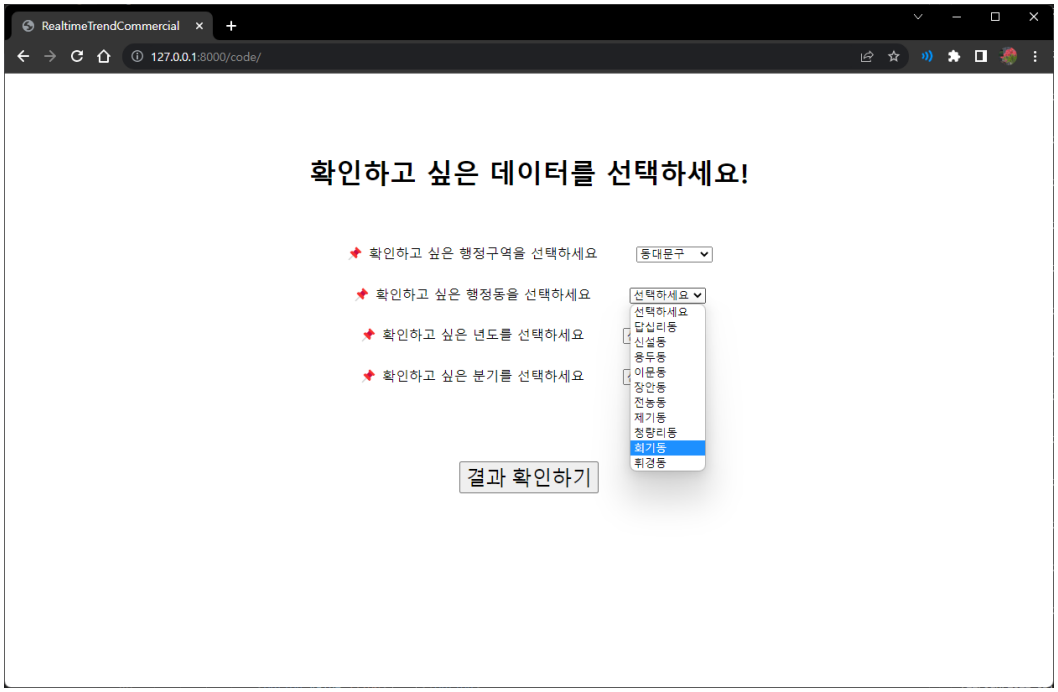


[그림 8-2] Spark Job 실행 시간 결과 그래프

4.3 프로젝트 결과

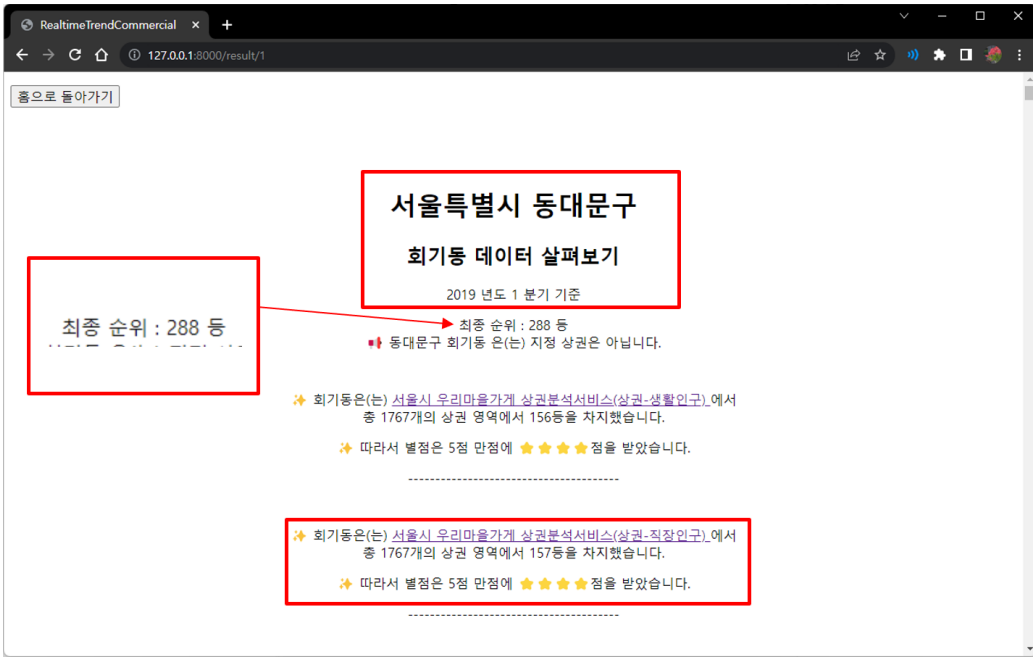
[그림 9-1] Django 첫 화면

사용자는 시작하기 버튼을 통해 데이터 분석 결과를 확인할 수 있고, 데이터 최신화하기 버튼을 누름으로써 Kafka producer 에게 요청을 보내 최신 데이터를 가져올 수 있다.



[그림 9-2] 사용자 선택 화면

사용자는 확인하고 싶은 행정구역과 행정동, 년도와 분기를 선택할 수 있다.



[그림 9-3] 결과 화면

10 개의 상권분석 서비스 분야에 대해서 사용자가 선택한 행정동이 약 1700 개의 서울시 행정동에 대해 그 분야에서 몇 위를 했고, 별 5 점 만점에 대해 몇 개의 별을 가져갔는 지에 따라 추천 정도를 확인할 수 있다. 상단에 최종 순위는 10 개의 분야에 대해 각각 얻은 순위를 통해

최종 순위를 확인할 수 있게 하여 사용자에게 사용자가 선택한 지역의 창업 추천 순위를 확인할 수 있다.

✨ 종로구 교남동은(는) 서울시 우리마을가게 상권분석서비스(상권-상권변화지표) 에서
총 1767개의 상권 영역에서 33등을 차지했습니다.

✨ 따라서 별점은 5점 만점에 ⭐⭐⭐⭐⭐점을 받았습니다.

✨ 종로구 무악동은(는) 서울시 우리마을가게 상권분석서비스(상권-생활인구) 에서
총 1767개의 상권 영역에서 383등을 차지했습니다.

✨ 따라서 별점은 5점 만점에 ⭐⭐⭐점을 받았습니다.

✨ 중구 황학동은(는) 서울시 우리마을가게 상권분석서비스(상권-점포) 에서
총 1767개의 상권 영역에서 87등을 차지했습니다.

✨ 따라서 별점은 5점 만점에 ⭐⭐⭐⭐⭐점을 받았습니다.

✨ 성동구 마장동은(는) 서울시 우리마을가게 상권분석서비스(상권-생활인구) 에서
총 1767개의 상권 영역에서 429등을 차지했습니다.

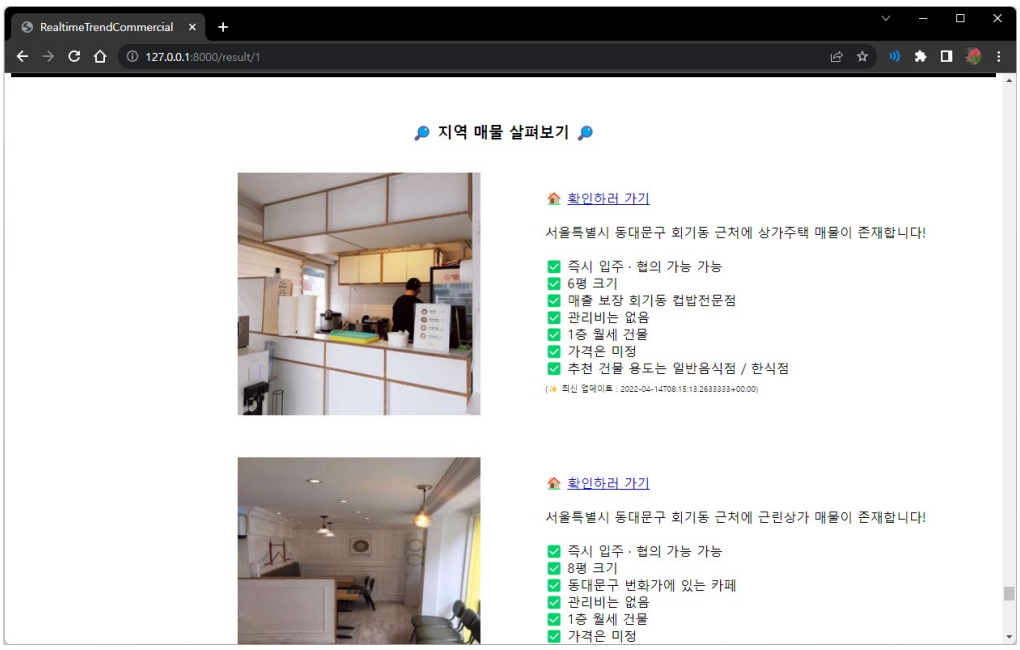
✨ 따라서 별점은 5점 만점에 ⭐⭐⭐점을 받았습니다.

✨ 광진구 군자동은(는) 서울시 우리마을가게 상권분석서비스(상권배후지-아파트) 에서
총 1767개의 상권 영역에서 440등을 차지했습니다.

✨ 따라서 별점은 5점 만점에 ⭐⭐⭐점을 받았습니다.

[그림 9-4] 결과 화면

또한 사용자가 선택한 행정동 근처 행정구역에 대해서도 각각 어느 분야에서 몇 순위를 했는지 정보를 보여주어 사용자가 선택한 행정구역의 상대적인 장점과 단점을 확인할 수 있게 하였다.



[그림 9-5] 매물 정보 화면

추가적으로 최종적으로 사용자가 선택한 지역에 등록된 매물 정보들을 확인할 수 있다. 매물 사진과 함께 매물 크기, 상세 위치, 관리비와 가격 정보를 알 수 있고 등록된 매물의 과거 정보를 토대로 추천 건물 용도를 확인할 수 있다.

5. 결론 및 기대효과

5.1 결론

최근 취업난으로 인해 많은 20, 30 대들이 취업을 하고자 한다. 하지만 그들은 아직 사회 초년생이라 부동산에 가서 어떤 것을 물어봐야하는지 모른다. 부동산업체들도 본인들의 생각에 따라 해당 위치의 상권을 말할 뿐, 현재 트렌드를 모두 알지는 못한다. 따라서 해당 앱의 도움을 받는다면 사용자가 원하는 위치에서 트렌디함과 동시에 객관적인 수치로 유동인구나 매출정보를 제공함으로써 효율적으로 업종을 고르며 위치를 선정할 수 있다. 창업 용도 외에도, 주위에 냄새가 강한 식당들이 많다면 개인 연습실이나 작은 사무실을 구하는 사람이 피할 수 있는 이점도 있다.

사용자가 여러 곳에서 얻어야 하는 정보를 프로젝트 서비스에서 단번에 확인할 수 있게 하여 편의성을 제공하고 많은 정보가 없는 초보 창업인들에게 필수적인 정보를 수치를 통해 직관적으로 보여줌으로써 창업의 어려움을 극복 가능하게 해준다.

본 서비스를 진행하며 대용량의 데이터를 각각의 사이트에서 별도로 가져오는데 많은 변수들이 있다는 사실을 알 수 있었다. 이러한 변수를 통제하기 위해 실험을 진행하며 어떤 설정 값이 본 서비스에 최적으로 동작하는지 알아낼 수 있었다. 비록 기업의 입장이 아니라 많은 제약 사항이 있었지만, 이를 대비하기 위해 본 서비스는 데이터를 최대한 잘게 잘라서 분산효과를 확대시켰다.

본 서비스의 가장 큰 목표는 수많은 리소스로부터 크기도 다양하고 형식도 다른 데이터를 최대한 유실없이 가져오고 본 서비스의 특색에 맞게 정제하여 사용자가 필요로 하는 결과값을 도출하는 것이었다. 앞서 언급한 연구 결과들을 토대로 Kafka 와 Spark 에 여러 환경 변수들을 주고 서비스 성능에 영향을 줄 수 있는 환경을 구성하여 기존의 설정 값보다 시간 단축도 하고 결과값도 안정적으로 전달할 수 있게 되었다.

5.2 추후 연구 방향

본 서비스는 데이터 수집의 한계로 서울의 특정 지역만 대상으로 진행했다. 하지만 사업적인 모델로 발전시켜 다양한 데이터를 수용할 수 있다면, 전국 단위로 확장 가능하다. 예를 들어,

전국 각지에 있는 CCTV 나 각 행정 자치부가 서로 협력해 비슷한 정보를 제공한다면 본 서비스는 일반화된 API 를 통해 데이터를 수집하기 때문에 언제든 적용 가능하다. 본 서비스는 단지 예비 창업자들만을 위한 서비스로 한계를 맞이하는 것이 아니라 금융, 토지, 의료 등 각종 분야에 필요한 데이터만 있다면 Kafka 와 Spark 를 주축으로 하여 데이터 수집 형태만을 변형시켜 적용 가능하다. 즉, 본 서비스는 각 기술들의 안정적인 연계를 일반화하는 방향으로 두어 어떤 분야들이든 손쉽게 발전시킬 수 있다.

6. 참고문헌

[1] 송준석 외 3 인, 제한된 메모리 환경에서의 아파치 스파크 성능 비교(2016)

[2] 정재화, Spark SQL 기반 고도 분석 지원 프레임워크 설계(2016)

[3] https://spark.apache.org/docs/latest/configuration.html

[4] Jeffrey Aven, Data Analytics with Spark Using Python (Addison-Wesley Data & Analytics Series) 1st Edition, ISBN-13: 978-0134846019

[5] How to send Large Messages in Apache Kafka, https://www.conduktor.io/kafka/how-to-send-large-messages-in-apache-kafka

[6] Kafka - Spark Streaming 1메가 이상의 메시지 처리 방법, https://nashorn.tistory.com/entry/Kafka-Spark-Streaming-1%EB%A9%94%EA%B0%80-%EC%9D%B4%EC%83%81%EC%9D%98-%EB%A9%94%EC%8B%9C%EC%A7%80-%EC%B2%98%EB%A6%AC-%EB%B0%A9%EB%B2%95

[7] Create an S3 sink connector by Confluent, https://developer.aiven.io/docs/products/kafka/kafka-connect/howto/s3-sink-connector-confluent

[8] New Apache Kafka to AWS S3 Connector, https://lenses.io/blog/2020/11/new-kafka-to-S3-connector/