碩士學位論文

Hadoop 기반 전기자동차 배터리 소모량 데이터의 정제 메카니즘 개발

濟州大學校 大學院

電算統計學科

金敬洹

2015年 8月

碩士學位論文

Hadoop 기반 전기자동차 배터리 소모량 데이터의 정제 메카니즘 개발

濟州大學校 大學院

電算統計學科

金敬洹

2015年 8月

Hadoop 기반 전기자동차 배터리 소모량 데이터의 정제 메카니즘 개발

指導交綏 朴 景 麟

金敬洹

이 論文을 理學 碩士學位 論文으로 提出함

2015年 6月

金敬洹의 理學 碩士學位 論文을 認准함

審査委員長 이 정 훈

委　　原 박 경 린

委　　原 송 준 모

濟州大學校 大學院

2015年 6月

Development of a data refinement mechanism for the battery consumption of electric vehicles based on Hadoop

Kyungwon Kim

(Supervised by professor Gyung-Leen Park)

A thesis submitted in partial fulfillment of the requirement for the degree of Master of Science

2015. 8.

Department of Computer Science and Statistics
GRADUATE SCHOOL
JEJU NATIONAL UNIVERSITY

목 차

List of Tables

List of Figures

Abstract

스마트그리드 5대 분야 중 지능형 운송에 해당하는 전기자동차는 국가적으로 연구개발 및 보급을 추진하고 있다. 제주대학교 전기차사업단이 총괄 주관하여 추진하였던 '2012년 광역경제권연계협력사업'에서 생성된 전기자동차 주행에 대한 원시데이터를 Hadoop을 사용하여 가공하고, 분석한다. 원시데이터는 1초마다 14개의 필드를 가지는 레코드로 구성되어 있고, 중복된 GPS 좌표 및 측정 센서의 오류값을 포함하고 있다. 또한, 동일구간에 대한 데이터를 비교해보았을 때 SoC 초기값이 다르게 나타나는 문제점이 있다. 이러한 원시데이터를 제주대학교 전기차사업단에서 관리하는 웹서버에 설치된 Hadoop과 Pig Script를 사용하여 중복된 좌표값을 제거하고 SoC에 대한 정규화 작업을 한다. Pig는 Map-Reduce를 대체하는 Pig Latin을 제공하며, 간단한 코딩으로 데이터의 가공이 가능하다. 또한, Hadoop Pig는 사용자 정의함수인 UDF(User Define Function)를 제공하며, 이 UDF를 사용하여 GPS 좌표값으로 누적 이동 거리를 계산한다.

정제된 데이터를 도식화하고 분석하여 오르막구간에서는 배터리가 지속적으로 소비되는 것과 내리막구간에서는 회생 제동으로 인하여 배터리 소비가 적은 것을 확인한다. 또한, 동일구간의 여러 데이터를 비교해 본 결과 0.158 : 0.061로 내리막구간에서는 오르막구간 보다 61%의 배터리 소비량이 줄어든 것을 알 수 있다. 동일지점에 대해 정방향과 역방향 주행데이터를 비교해 본 결과 회생 제동으로 충전되는 배터리 양이 0.2 : 0.227로 소비되는 양보다 높게 나타났으며, 이는 동일구간을 왕복 주행 하더라도 SoC의 소비량은 다르지만, 정규화를 통해 0.0 ~ 1.0 사이로 변환하였기에 생기는 오차로 예상된다. 도식화한 데이터를 바탕으로 전기자동차는 운전자의 운전습관, 회생 제동, 주변 환경요인에 따라 SoC 소비량에 변화가 있음을 확인한다.

전기자동차의 보급이 더욱 활성화되면 방대한 양의 데이터 스트림이 생성될 것이며, 제안 메카니즘을 통해 쉬운 분석이 가능할 것이다. 그리고 다른 스마트그리드 엔티티에서 생성되는 데이터 스트림에도 적용 가능할 것으로 예상한다.

Abstract

An EV (Electric Vehicle) is the most essential element in the smart grid transportation, which is one of the 5 core smart grid areas in the Republic of Korea. While the national government is accelerating the penetration of EVs and also the development of EV-related business models, JNU (Jeju National University) EVRC (Electric Vehicle Research Center) has carried out a inter-metropolitan area cooperative research project in which a tracking system acquires and accumulates the SoC (State-Of-Charge) stream from each EV. Here, an EV reports its status record consisting of 14 fields, definitely including spatial and temporal stamps, to the central server with a period of 1 second.

According to our observation, the record stream contains duplicated GPS coordinates and erroneous sensor readings. Moreover, each trip begins from different SoC levels, leading to different consumption patterns. To solve this problem, this thesis develops a Pig script to remove redundant information and normalize SoC values, taking advantage of the Hadoop platform installed in the JNU EVRC web server. Next, a user-defined function, or UDF in short, is implemented to convert the record-by-record location to the accumulated distance from the start point.

After the visualization of the filtered stream, our analysis finds that the SoC level almost linearly drops along the upslope area, while it increases in the downhill road due to the regenerative braking torque. In addition, the comparative analysis for 4 different trips on a single route, reveals the SoC consumption in the downslope is less than that in the upslope by 61 %. Next, for bidirectional trips between the two same points, the amount of battery charged by the regenerative torque is larger than that consumed by 02.:0.227.

This result confirms the effect of drivers' behavior, regenerative brake, and other environmental aspect to the energy consumption of EVs.

After all, it is expected that a large volume of data streams will be created according to the deployment of EVs, and our mechanism will make it possible to systematically analyze the big data. Moreover, this framework can apply to other smart grid entity streams and integrate heterogeneous ones.

I. 서론

스마트그리드는 “스마트(smart)와 그리드(grid)”의 합성어로 여기서 스마트는 IT 기술을 그리드는 전력망을 뜻하는데, 결국 기존 전력망에 IT 기술을 접목시켜 소비자와 전력공급자 사이에 양방향, 실시간으로 정보 교환을 함으로써 에너지 효율을 최적화하는 지능형 전력망을 의미한다. 전력공급자는 전력수급현황에 따라 탄력적으로 가격을 설정하여 공급할 수 있으며, 소비자는 실시간으로 사용전력량 및 전력 가격을 확인할 수 있어 효율적인 전력소비를 가능하게 한다. 국내의 경우에도 스마트그리드 5대 분야를 선정하여 개발 및 구축, 실증하고 있다. 1) 기존의 전력망에 IT 기술을 접목하여 신뢰도 및 운용 효율을 향상 시키는 지능형 전력망, 2) 양방향 통신 인프라를 접목하여 소비자에게 다양한 서비스를 제공함으로써 에너지 효율을 향상하는 지능형 소비자, 3) 다양한 신재생 에너지의 효율적인 운영을 할 수 있게 하는 지능형 신재생, 4) 시간대에 따라 변동하는 전력가치를 IT 기술을 기반으로 실시간으로 전력 소비자에게 제공함으로써 소비자로 하여금 실시간 전력가격에 스마트하게 반응하여 자신의 전력소비를 조정하도록 하는 가상발전기, 수요반응, V2G 등 다양한 새로운 스마트그리드 자원을 이용하여 효율적인 전력망의 이용을 가능토록 하는 지능형 전력서비스, 5) 탄소 발생 억제가 가능한 전기자동차의 확대 보급의 기반을 마련하는 지능형 운송을 포함한다[1].

전기자동차(Electric Vehicle)는 일반적으로 전기에너지를 배터리에 저장하고 배터리로부터 구동력을 얻어 움직이는 자동차를 의미한다[2]. 전기자동차는 현재 이슈가 되고 있는 지구온난화, 이산화탄소 배출량 규제 강화, 화석연료 고갈 및 고유가 문제 등 이를 해결하기 위한 긍정적인 효과를 가져다 줄 수 있어, 전기자동차 보급 및 연구개발이 최근 들어 활발히 진행되고 있다[1]. 더욱이 일반 내연기관 자동차보다 유지비가 적게 들고 소음이 없으며, 유지보수가 간편한 장점이 있다. 그러나 근래의

배터리 기술의 발전에도 불구하고 1회 충전 시 주행거리가 150km 이하로 내연기관 자동차 대비 짧은 단점이 있다. 이에 전기자동차 보급을 촉진하기 위해서는 충전 인프라의 구축 및 주행거리를 증가시키기 위한 연구개발이 필요하다.
우리나라에서는 산업통상자원부에서 기술개발을 지원하고 환경부는 친환경차 보급을 총괄한다. 국토교통부는 도로교통제도, 기획재정부는 세금감면 등의 정책으로 지원하고 있다. 제주특별자치도는 '카본 프리 아일랜드 2030(Carbon Free Island Jeju by 2030, CFI2030)'의 정책으로 2030년까지 제주도 내 전기자동차 100% 보급을 목표로 하여 전기자동차 보급에 적극적인 자세를 보이고 있다. 제주대학교 내에서도 활발한 연구 개발이 진행 중이며, 대표적으로 산업통상자원부가 지원하는 「광역경제권연계협력사업」 '전기차 산업 활성화를 위한 차량 부품 및 운영시스템 개발 ' 과제에 주도적으로 참여하고 있다. 위의 과제는 제주대학교 전기차사업단이 총괄 주관기관을 맡고 있으며, 1세부과제에서는 전기차 관제시스템, 2세부과제에서는 신재생 연계가 가능한 충전기의 개발, 3세부과제에서는 전기자동차 전용 내비게이션 개발 등을 주 내용으로 삼고 있다[3].

전기자동차의 활발한 보급을 위해서는 충전 인프라의 구축 및 현재 충전된 배터리로 주행거리를 최대한 연장하는 것이 필요하다. 제주도는 '제주 스마트그리드 실증사업'을 통하여 충전 인프라의 기반이 조성되어 있으며 한번 충전된 배터리의 주행거리를 연장하기 위해 연구개발을 지원하고 있다. 한번 충전된 배터리의 주행거리를 연장하기 위해서 본 논문에서는 전기자동차의 주행 데이터를 가공하여 정밀분석이 쉽도록 하는 전처리 작업이 중요하다. 본 논문에서 사용되는 전기자동차 주행 데이터는 위에서 언급한 광역경제권연계협력사업의 3세부과제의 전기자동차 주행 데이터를 사용하여 진행한다. 전기자동차 1대의 주행에 대하여 1초마다 하나의 레코드가 생성되는데, 이 레코드에는 위도, 경도, 셀리지온 위도, 셀리지온 경도, GPS고도, 속도, 온도, 습도, 압력, 고도, SoC(State of Charge, 배터리 잔량), 시간, 기울기1, 기울기2 등의 필드들이 포함된다. 시간에 따라 계속적으로 축적되는 방대한 양의 데이터를 분석하기 위해서는 빅데이터 처리기술이 필요하며, 입력데이터를 분석하기 쉬운 데이터로 처리하는 분석 메카니즘이 필요하다.

빅데이터 처리 기술로는 Hadoop, NoSQL, R 등이 있으며, 그 중에서도 가장 대표적인 Hadoop은 현재 정형 및 비정형 빅데이터 분석에서 가장 선호되는 솔루션이다. HDFS(Hadoop Distributed File System)에 파일을 저장하고, Map-Reduce 방식으로 많은 양의 데이터를 빠른 속도로 분산 병렬 처리한다[4]. Hadoop은 다양한 서브프로젝트를 제공하며 그 중 Hive 와 Pig가 대표적이다. Hive는 데이터 웨어하우징용 솔루션이며, HiveQL이라는 SQL과 유사한 쿼리를 사용하여 HiveQL로 정의한 내용을 Hive가 Map-Reduce Job으로 변환하여 실행한다. 반면 Pig는 아파치 프로젝트에 속해있으며 복잡한 Map-Reduce 프로그래밍을 대체할 Pig Latin이라는 자체 언어를 제공하여 Map-Reduce API를 매우 단순화시키는 한편 데이터 플로우 형태의 프로그램 인터페이스를 제공한다. 그리고 다중값과 중첩된 형태를 보이는 좀 더 다양한 데이터 구조를 지원한다.

본 논문에서는 Hadoop Pig를 사용하여 구현된 분석 메카니즘을 제안하며, 본 논문의 구성은 다음과 같다. 1장에서 논문에서 해결하고자 하는 문제를 개관하고 2장에서는 배경과 관련 연구들을 조사한다. 3장에서는 제안하는 기법을 구축하기 위한 Hadoop Pig의 인터페이스를 기술하고 Pig UDF(User Define Function)를 이용해 구현된 분석 플랫폼을 확장한다. 4장에서는 제안된 기법을 통하여 정제된 데이터에 대해 자세히 설명한다. 최종적으로 5장에서는 본 논문을 요약하고 결론을 도출한다.

II. 배경과 관련 연구

1. 「광역경제권연계협력사업」 전기차 산업 활성화를 위한 차량부품 및 운영시스템 개발 사업의 수행.

제주대학교 전기차사업단에서는 「2012년 광역경제권연계협력사업」을 수행하였다. 제주권과 전남권이 연계 협력하여 수행하였고 과제명은 '전기차 산업 활성화를 위한 차량 부품 및 운영시스템 개발'이며 총 연구개발기간은 총 3년으로 되어있다. 제주대학교 전기차사업단이 총괄 주관기관을 맡고 있으며 9개의 도내·외 기업이 사업에 참여하였으며, 과제는 총 3개의 세부과제로 구성되어 있다. 1세부과제는 전기자동차 통합관제시스템 개발을 목표로 제주권 기업이 맡았으며, 2세부과제는 신재생에너지 연계형 양방향 V2G/V2H 충전기 개발을 목표로 전남권 기업이 맡았다. 마지막으로 3세부 과제에서는 전기자동차 특화 내비게이션 개발을 목표로 제주권 기업에서 맡아 기술개발을 진행하였다. 전남권의 신재생에너지 및 충전기 제조기술과 제주권의 전기자동차 인프라를 활용하여 전기자동차 운영 활성화를 위한 기술개발이 주된 내용이다.

1세부 과제의 과제명은 '전기차 운용 관제시스템 및 시뮬레이터 개발'이며, 제주권 기업이 주도하여 연구개발을 진행하였다. 전기자동차가 다양한 산업에 활용될 수 있도록 모듈화된 시스템으로 관제시스템 프레임워크를 설계하며, 또한, EV 택시, EV 카쉐어링, EV렌터카 시스템 도입 시 예상되는 투자 대비 효과를 분석할 수 있는 도구를 개발하여 전기자동차 비즈니스 모델의 경제성을 분석하였다.

2세부 과제의 과제명은 '이종 전기차 스마트 충전시스템 및 하이브리드 일체형 전력모듈 개발'이며, 충전기의 고도화를 통하여 충전부문에서 선도적인 모델을 개발하고 실증했다. 제주도 내 신재생 기반 전기자동차 충전 인프라에 대한 시설 및 실증을

통한 효과성을 입증하였으며, 특히 양방향 V2G/V2H 충전기는 실시간 전력요금제에 대비하여 전력을 싸게 충전하고 비싸게 역송전하는 사업 모델로 전원이 없는 지역에서도 간편하게 충전기를 시설할 수 있도록 산업 적용 영역을 다변화하였다.
3세부 과제의 과제명은 '전기차 운영 서비스 기술개발' 이며, 충전 정보를 연계한 전기자동차 내비게이션 및 도로의 상황을 고려한 전기자동차에 특화된 지도를 자동 생성하는 도로 상황 인지형 로드맵퍼(Road Mapper)를 제작하여 전기자동차에 특화된 서비스를 제공하였다.
전기자동차 비즈니스 모델 개발 (전기차 운용 관제 시스템 및 시뮬레이터 개발), 충전 시스템 고도화 및 실용화 (전기차 스마트 충전 시스템 및 하이브리드 일체형 전력 모듈 개발), 전기자동차 운영 서비스 기술 개발 (전기차 특화 텔레매틱스 서비스 개발)로 전기자동차 활용 비즈니스에 쉽게 접근하며 전기자동차 산업의 활성화를 목표로 하고 있다. 이 광역경제권연계협력사업은 그 개발 과정과 제품 운영 중에 전기자동차 배터리소모, 충전소 상태 모니터링 등 다양한 고부가가치의 대용량 데이터를 생산하고 있다[3].

2. Hadoop

Hadoop은 분산파일시스템인 HDFS(Hadoop Distributed File System)와 Map-Reduce를 사용하여 대용량 데이터를 분산 처리 할 수 있는 자바 기반의 오픈소스 프레임워크이며, 기성 하드웨어를 사용할 수 있게 디자인 되어있다[4][5]. HDFS에 데이터를 저장하고, Map-Reduce를 이용해 데이터를 처리한다. Hadoop은 2003년 구글에서 발표한 GFS(Google File System)와 Map-Reduce를 구현한 결과물이다[6]. 초기 개발 당시 너치(Nutch)에 적용하기 위한 아파치 루씬(Lucene)의 하부 프로젝트로 시작되었으나, 2008년에는 아파치 최상위 프로젝트로 독립되었다[6][8]. Hadoop에서 HDFS와 Map-Reduce는 물리적으로 하나의 서버에 존재한다. Hadoop에서 분산처리를 위해 하나의 마스터 노드와 다수의 슬레이브 노드를 사용할 수 있다[5]. 마스터 노드는 파일시스템을 관리하기 위한 네임노드(Name node)와 네임노드에서의 장애나 데이터 손실을 줄이기 위한 이차 네임노드(Secondary Name

node), 작업 스케쥴링을 관리하기 위한 Job Tracker로 구성되어있다. 슬레이브 노드는 파일을 저장하기 위한 데이터노드 (Data node), 작업을 실행하기 위한 Task Tracker로 구성되어 있다. <그림1>은 Hadoop의 구성을 나타낸 것이다[9].

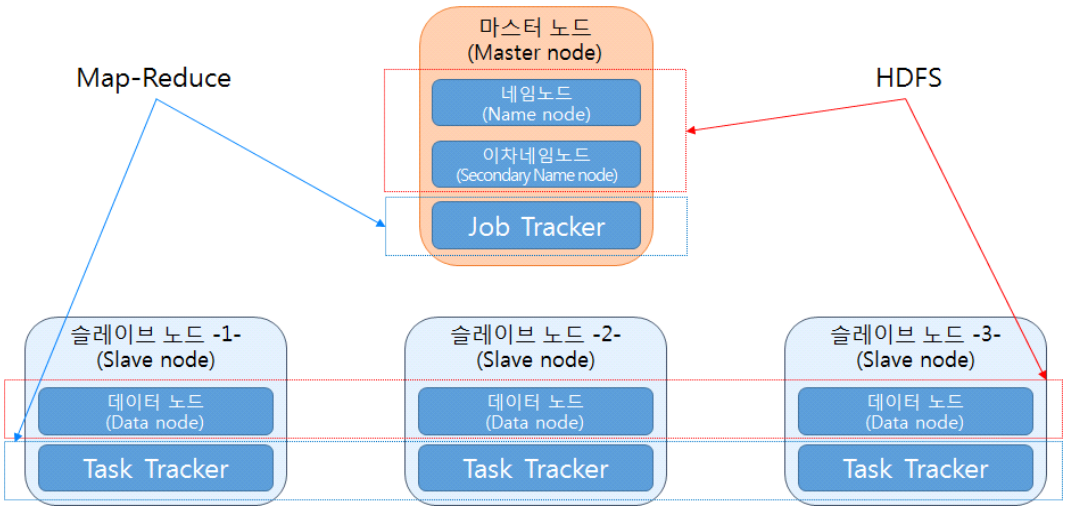


그림 1. Hadoop의 구성

2.1 HDFS

HDFS는 대용량 파일의 처리를 위한 자바로 개발된 분산 파일 시스템이다. 여러 노드의 디스크 공간을 활용하여 하나의 거대한 파일시스템을 구성한다. HDFS의 구성은 하나의 마스터 노드와 다수의 슬레이브 노드로 구성되어진다. 마스터노드는 네임노드, 이차 네임노드로 구성되며, 슬레이브 노드는 데이터노드로 구성된다. HDFS에 저장하는 파일은 특정 사이즈의 블록으로 나눠서 분산된 서버에 저장되며, 블록의 사이즈는 설정을 통해 변경할 수 있다. 기본적으로 하나의 블록은 64MB로 설정되어 있고, 복제본을 3개씩 저장 한다[6]. HDFS는 노드의 확장이 쉽기 때문에 초기 시스템 구축 시 데이터 증가를 예상하여 큰 저장소를 구현할 필요가 없이 상황에 따라 확장할 수 있다. 또한, 노드의 장애로 인한 데이터의 손실을 방지할 수 있게 설계되어있다.

2.2 Map-Reduce

Hadoop의 Map-Reduce는 방대한 양의 데이터를 병렬로 처리하게 해주는 분산 프로그래밍 모델이다[6]. Map-Reduce 프로그래밍 모델은 Map과 Reduce 두 개의 함수로 구성된다. 여기서 Map은 흩어져 있는 데이터 중 관련 있는 데이터끼리 묶는 작업을 통해서 임시 데이터 집합으로 만드는 역할을 하며 Reduce는 Map 작업에서 생성된 임시 데이터 집합에서 중복 데이터를 제거하고 원하는 데이터를 추출하는 작업을 담당한다.

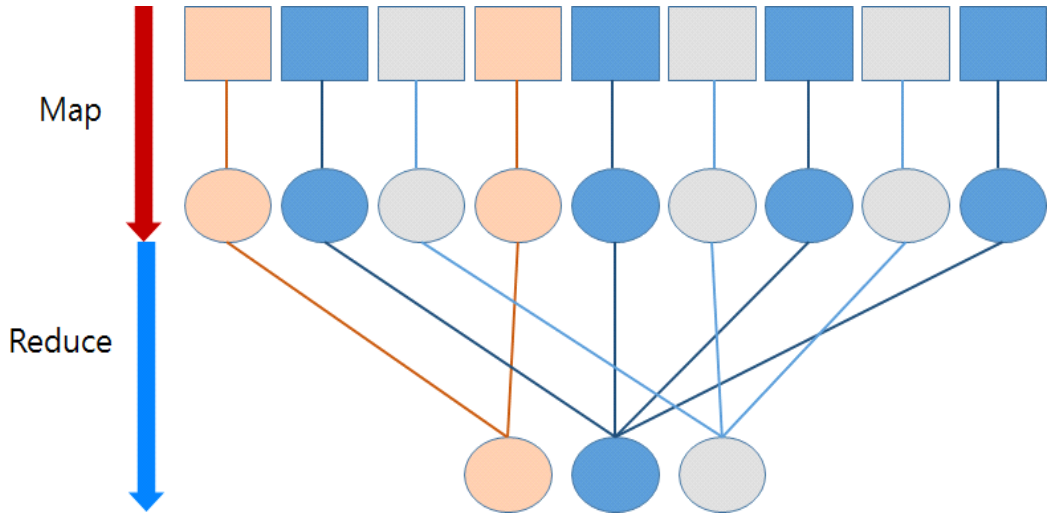


그림 2. Map-Reduce의 처리구조

Map-Reduce의 동작은 입력 데이터 셋에서 Key와 Value를 쌍으로 묶어서 Map 함수의 입력 값으로 보내는 작업으로 시작한다. 그 후 Map 함수를 통하면 필요한 분석대상만 추출 가능하며, Key와 Value를 리스트로 정렬, 그룹핑을 하여 Reduce 함수의 입력값으로 보내면 Reduce 함수를 통하여 원하는 결과값을 얻을 수 있다[10].

2.3 Hadoop ECO System

Hadoop은 빅 데이터를 분석, 처리하기 위해 사용되며 정형, 비정형 데이터도 처리하여야 하기에 다양한 기술들이 필요하다[7]. Hadoop 프레임워크를 이루는 다양한 서

브프로젝트들이 제공되며 이 서브프로젝트의 모임을 Hadoop ECO system이라 한다. Hadoop ECO system에서 HDFS와 Map-Reduce가 코어 프로젝트에 해당하며 나머지 프로젝트는 Hadoop의 서브 프로젝트이다. Hadoop 코어 프로젝트는 대용량 데이터의 분산 저장 및 처리를 담당하며 서브 프로젝트는 데이터 마이닝, 분석 등을 수행한다. <그림4>는 Hadoop ECO System을 나타낸 것이며 서브프로젝트의 내용은 <표1>을 참고하면 된다[11].

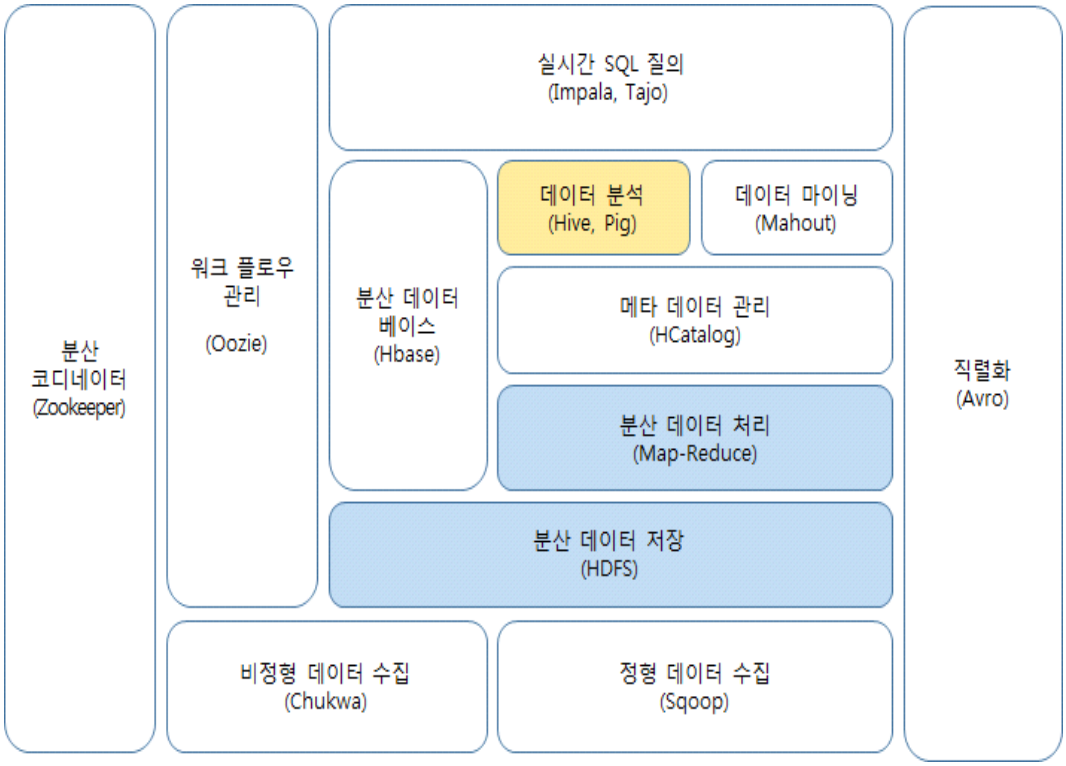


그림 3. Hadoop ECO System

표 1. Hadoop ECO System의 서브프로젝트[12].

명칭	설명
ZooKeeper	분산 처리 환경에서 정보 공유, 락 등 보조 기능으로 서버들 간의 시스템을 관리
Ooozie	Hadoop 작업을 관리하는 워크플로우 및 코디네이터 시스템
HBase	실시간 랜덤 조회 및 업데이트가 가능한 HDFS의 칼럼 기반 분산 데이터베이스
Hive	Hive QL 언어를 사용하여 데이터 쿼리를 수행하는 Hadoop 기반의 데이터 웨어 하우징용 솔루션
Pig	복잡한 Map-Reduce 프로그래밍을 대체할 Pig Latin이라는 자체 언어를 사용하는 스크립트 언어
Mahout	Hadoop 기반의 데이터 마이닝 알고리즘을 구현한 오픈 소스로 분류, 필터링, 패턴 마이닝 등의 알고리즘을 지원
HCatalog	Hadoop 데이터용 테이블 및 스토리지 관리 서비스
Avro	이기종 간 데이터 타입을 교환할 수 있는 체계를 제공
Chukwa	데이터를 HDFS에 안정적으로 저장시키는 플랫폼으로 에이전트와 콜렉터로 구성됨
Sqoop	HDFS, RDBMS, DW, NoSQL 등 저장소에 대용량 데이터를 전송할 수 있는 방법을 제공하는 대용량 데이터 전송 솔루션
Impala	Map-Reduce를 사용하지 않고, 자체 개발한 엔진을 사용하는 Hadoop 기반의 실시간 SQL 질의 시스템
Tajo	HDFS를 사용하며, SQL을 통해 실시간으로 데이터를 조회할 수 있는 Hadoop기반 DW시스템

3. 스마트그리드 관련 빅데이터 분석 사례

3.1 국내

제주대학교는 'Data analysis framework for charging facility' 논문에서 전기자동차 충전 설비의 모니터링을 통해 실시간으로 수집되는 방대한 양의 스트림의 분석하였다. 이 논문은, 원시 스트림 데이터를 Hadoop 기반의 프레임워크를 기반으로 변환하여 필요한 정보 필드를 필터링하였으며, 기본적인 분석을 하였다. 분석을 통하여 개인 소유 차량의 위치 등에 따라 각 전기자동차 엔티티 사이의 유의한 차이를 발견하였다.

'Encored'사는 가정용과 상업용으로 나누어서 '에너지 플래너' 서비스를 하고 있다. '에너지 플래너' 서비스는 'Encored'사에서 개량한 AMI(Advanced Metering Infrastructure, 첨단계량인프라)에서 1초마다 수집되는 전력 빅데이터를 분석하여 사용자에게 전기 소비패턴 안내, 전력 이상 사용 시 알림서비스, 기기별 수요 예측, 기기 이상 징후 포착 등의 정보를 제공한다[13].

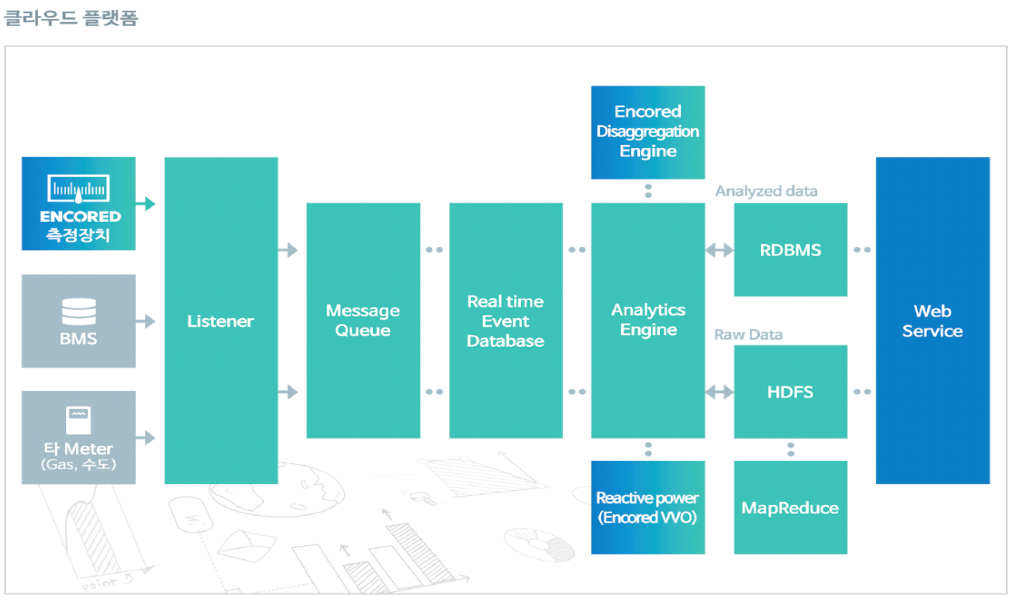


그림 4. 'Encored'사의 클라우드 플랫폼

‘한전 KDN’에서는 전력 분야의 전 영역에 대하여 빅데이터 기반의 비즈니스 모델 발굴에 적극적으로 나서고 있다. 스마트그리드 환경에서는 SCADA(Supervisory Control And Data Acquisition, 원방감시제어시스템), DAS(Distribution Automation System, 배전자동화시스템), AMI 등 실시간 전력정보와 신재생에너지, 에너지저장장치(ESS) 등 다양한 분산전원의 계통운영정보에서 대량의 정보가 발생하고 있다. 이 대량의 정보를 빅데이터 기반의 ICT 융합기술을 활용해 경제적인 계통운영을 하고자 한다. ‘한전 KDN’은 한전에서 성과발표회를 한 스마트그리드 종합운영시스템의 개발 초기부터 참여하고 있으며, 이 시스템은 송·변전, 배전, 영업 등 기존 14종류의 업무영역별 운영시스템의 정보를 종합해 문제해결형 서비스를 제공하는 시스템을 말한다. 이 시스템의 도입으로 설비 이용률 10%, 실시간 통합운영정보 제공으로 인한 업무효율 30%가 향상될 것으로 예상한다[14].
그러나 국내의 경우 전력 관련 빅데이터를 가장 많이 소유한 한국전력이 보안 등의 이유로 공개를 꺼려 전력 빅데이터 사용이 활발하지 않은 실정이다. 그러나 최근 정부 3.0이 시작된 이후 산업통상자원부과 한국전력 사이에 빅데이터 공개를 두고 논의가 이루어지고 있으므로 빅데이터가 공개가 시작되면 새로운 산업이 열릴 것으로 생각된다[15].

3.2 국외

미국에서는 ‘그린버튼 이니셔티브’라는 전력관련 빅데이터 공개 제도를 통한 에너지 컨설팅 사업이 진행 중이다. 소비자가 웹상에서 에너지 사용량 등을 확인하고 스마트폰이나 PC를 통해 공유 할 수 있는 시스템이다. 미국 캘리포니아 주에서는 위 제도를 통하여 연간 1,500만 kW정도의 전기를 절감하는 효과를 얻기도 하였다[15].

영국 최대의 전기 및 가스회사인 ‘Centrica’사는 AMI를 통해 30분 단위 에너지 소비량에 대한 빅데이터를 수집하였다. ‘Centrica’사는 이 데이터를 활용하여 피크시간대의 실시간 전력수요 동향 분석, 시간대와 전력수요에 따라 동적으로 변하는 전기요금 설계, 이에 근거한 전력수요 관리 및 사용시간대 분산 등에 활용하고 있다. 이를 소비자의 그룹화, 요금제 개발, 소비예측 등에 활용하고 있으며, 소비자는 사용량의

실시간 확인 및 지난 사용량과의 비교분석을 통하여 연간 최대 190파운드의 에너지 비용을 절감하고 있다[15].

미국의 'OPower'사는 날씨, 전력소비 패턴 등의 정보를 분석하여 소비자에게 제공하고 있으며, 이 데이터를 분석하기 위해 Hadoop 기반의 분석 플랫폼을 구축하였다. 'OPower' 사는 분석한 정보를 우편, e-mail, SNS를 통하여 소비자에게 전달하며, 이를 통해 소비자는 평균 1.5~3.5%의 전기요금 절약 효과를 거두고 있다[16].

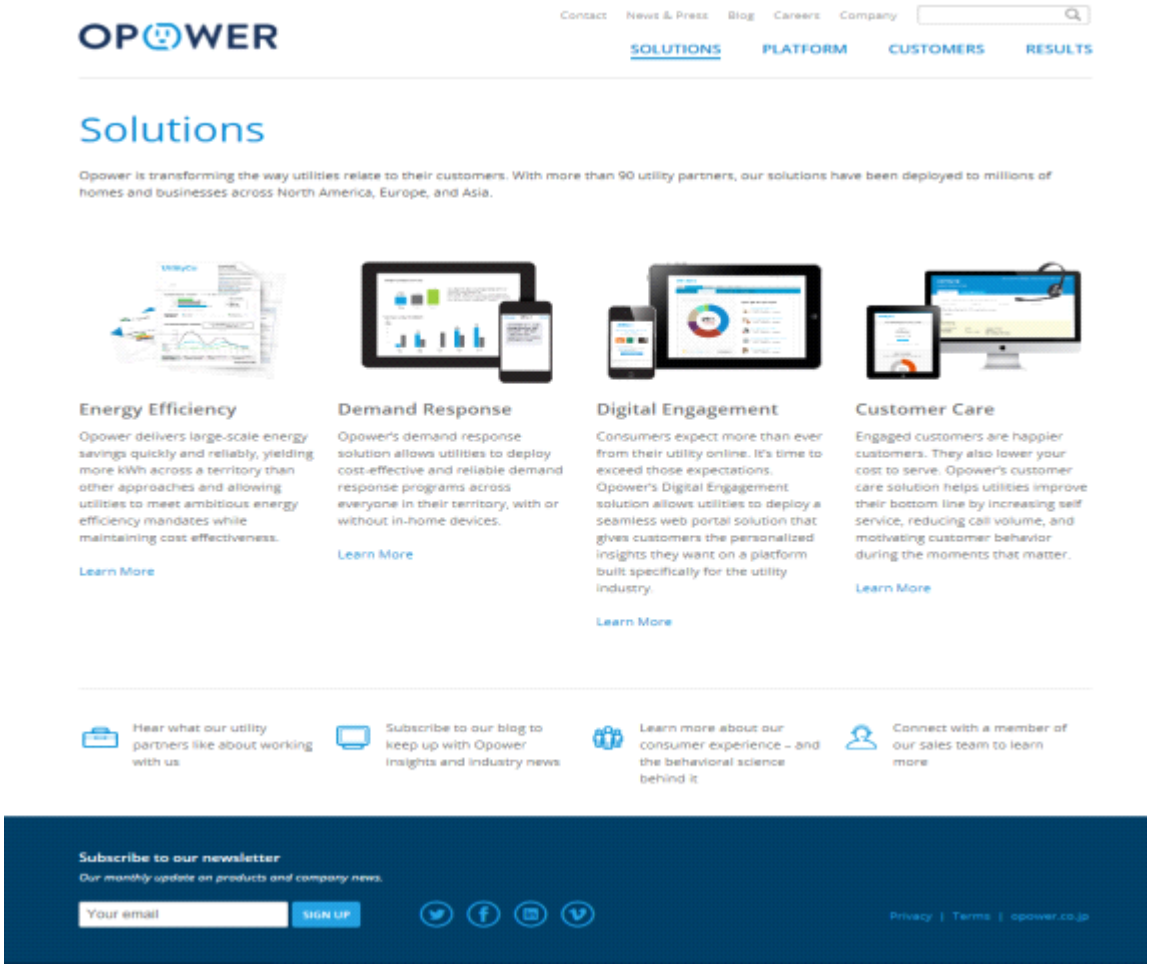


그림 5. 'OPower'사의 스마트그리드 솔루션[http://www.opower.com/]

III. 데이터 정제 메카니즘 분석

1. 원시 데이터 현황과 문제점

본 논문에서 사용한 원시 데이터는 '광역경제권연계협력사업'의 3세부과제에서 취득하였다. 시판 중인 전기자동차에 연구개발을 통하여 개발한 DCC(Driving Condition Collector)장치를 장착하여 주행 중 GPS(Global Positioning System) 좌표, 온도, 습도, 기울기, SoC 등을 1초 주기로 수집한 것으로, 측정 거리는 제주도 내 제주대학교 사거리에서 돈내코 입구 삼거리까지(516도로) 약 26km이다.

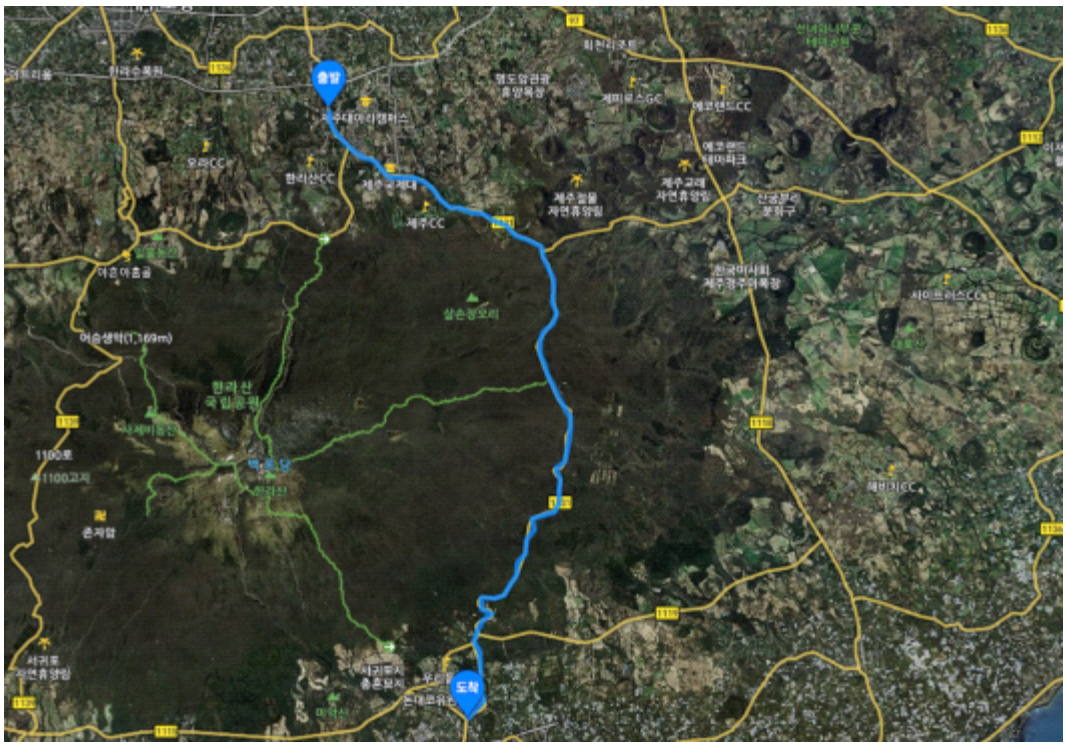


그림 6. 원시 데이터의 현황(이동구간)

	경도	위도	샐리지온경도	샐리지온위도	GPS고도	속도	온도	습도	압력	고도	배터리	시간	기울기1	기울기2
2	126.5504	33.4549	126.5507745	33.4567423	-210	2	11.5	55.1	997.999	703.12	64.5	2014-01-18 오후 8:44:16	0	0
3	126.5504	33.4549	126.5507745	33.4567423	-210	2	11.5	55.1	997.999	703.13	64.5	2014-01-18 오후 8:44:17	-0.1	0
4	126.5504	33.4549	126.5507745	33.4567423	-209.8	2	11.5	55.1	998.047	702.73	64.5	2014-01-18 오후 8:44:18	-0.2	0
5	126.5504	33.4549	126.5507745	33.4567423	-209.7	2	11.5	55.1	998.018	703	64.5	2014-01-18 오후 8:44:19	-0.2	0
6	126.5504	33.4549	126.5507745	33.4567423	-209.6	1	11.5	55.1	998.042	702.79	64.5	2014-01-18 오후 8:44:20	-0.2	0
7	126.5504	33.4549	126.5507745	33.4567423	-209.6	2	11.5	55.1	998.002	703.15	64.5	2014-01-18 오후 8:44:21	0.6	0
8	126.5504	33.4549	126.5507745	33.4567423	-209.6	2	11.5	55.1	998.012	703.07	64.5	2014-01-18 오후 8:44:22	9.6	5.2
9	126.5504	33.45489	126.5507745	33.4567423	-209.5	2	11.5	55.1	998.027	702.95	64.5	2014-01-18 오후 8:44:23	9.6	0
10	126.5504	33.45488	126.5507745	33.4567423	-209.5	5	11.5	55.1	998.012	703.1	64.5	2014-01-18 오후 8:44:24	10.3	0
11	126.5504	33.45487	126.5507771	33.4567448	-209.5	7	11.5	55.1	997.939	703.73	64.5	2014-01-18 오후 8:44:25	10.3	0
12	126.5504	33.45485	126.5507988	33.4567211	-209.5	10	11.4	55.1	997.901	704.04	64.5	2014-01-18 오후 8:44:26	4.6	0
13	126.5504	33.45485	126.55083	33.4566828	-209.5	10	11.4	55.1	997.884	704.16	64.5	2014-01-18 오후 8:44:27	4.6	0.4
14	126.5505	33.45476	126.5508611	33.4566286	-209.4	20	11.5	55.1	997.869	704.3	64.5	2014-01-18 오후 8:44:28	-11.3	-0.9
15	126.5505	33.4547	126.5508858	33.4565635	-209.5	22	11.5	55.1	997.908	704	64.5	2014-01-18 오후 8:44:29	-11.3	0
16	126.5505	33.45465	126.5509071	33.4564985	-209.5	21	11.5	55.1	997.896	704.1	64.5	2014-01-18 오후 8:44:30	-11.5	0
17	126.5505	33.45461	126.5509215	33.4564443	-209.5	18	11.5	55.1	997.924	703.88	64.5	2014-01-18 오후 8:44:31	-11.5	0
18	126.5505	33.45461	126.5509316	33.4564033	-209.5	18	11.5	55.1	997.918	703.94	64.5	2014-01-18 오후 8:44:32	-2.3	0
19	126.5505	33.45455	126.5509385	33.4563758	-209.5	12	11.5	55.1	997.894	704.16	64.5	2014-01-18 오후 8:44:33	-2.3	0
20	126.5505	33.45454	126.5509436	33.4563581	-209.5	10	11.5	55.5	997.901	704.1	64.5	2014-01-18 오후 8:44:34	-5.7	-2.6
21	126.5505	33.45452	126.5509468	33.456346	-209.6	7	11.5	55.5	997.867	704.39	64.5	2014-01-18 오후 8:44:35	-5.7	-2.6
22	126.5505	33.45452	126.5509494	33.4563389	-209.6	5	11.5	55.5	997.92	703.96	64.5	2014-01-18 오후 8:44:36	0.2	-2.6
⋮														
1843	126.5954	33.29596	126.5954758	33.296055	212.7	23	12	50.5	1004.759	648.95	40.5	2014-01-18 오후 9:15:04	-18	-3
1844	126.5953	33.29594	126.5954758	33.296055	212.2	18	12	50.5	1004.759	648.95	40.5	2014-01-18 오후 9:15:05	-20.3	-12.7
1845	126.5953	33.29592	126.5954679	33.2960385	211.7	13	12	50.5	1004.8	648.6	40.5	2014-01-18 오후 9:15:06	-20.3	-12.7
1846	126.5953	33.29592	126.5954678	33.2960323	211.7	13	12	50.5	1004.781	648.77	40.5	2014-01-18 오후 9:15:07	-12.3	-12.7
1847	126.5953	33.29592	126.5954698	33.2960333	210.9	5	12	50.5	1004.789	648.71	40.5	2014-01-18 오후 9:15:08	-12.3	-12.7
1848	126.5953	33.29592	126.5954715	33.2960338	210.4	3	12	50.5	1004.777	648.8	40.5	2014-01-18 오후 9:15:09	-12.2	-12.7
1849	126.5953	33.29592	126.5954715	33.2960338	210	2	12	50.5	1004.774	648.84	40.5	2014-01-18 오후 9:15:10	-12.2	-24.3
1850	126.5953	33.29593	126.5954715	33.2960338	209.5	1	12	50.5	1004.788	648.72	40.5	2014-01-18 오후 9:15:11	-12.4	-24.3
1851	126.5953	33.29593	126.5954715	33.2960338	209	1	12	50.5	1004.759	648.96	40.5	2014-01-18 오후 9:15:12	-12.4	-24.3
1852	126.5953	33.29593	126.5954715	33.2960338	208.6	0	12	50.5	1004.741	649.12	40.5	2014-01-18 오후 9:15:13	-12.4	-24.3
1853	126.5953	33.29593	126.5954715	33.2960338	208.3	1	12	50.5	1004.761	648.96	40.5	2014-01-18 오후 9:15:14	-12.4	-24.3
1854	126.5953	33.29593	126.5954715	33.2960338	208.1	2	12	50.5	1004.745	649.08	40.5	2014-01-18 오후 9:15:15	-12.3	-15.1
1855	126.5953	33.29594	126.5954715	33.2960338	207.8	2	12	50.5	1004.772	648.87	40.5	2014-01-18 오후 9:15:16	-12.3	

그림 7. 원시 데이터의 현황(Excel 예시)

이렇게 수집된 원시 데이터를 살펴보면 총 14개의 필드를 가지는 레코드가 1초마다 쌓이고 있는데, GPS 좌표, 고도, 기울기는 두 개의 필드로 존재한다. 그 이유는 GPS 좌표의 경우 DCC 장비 뿐만 아니라 배터리의 잔량 수집 장비에서 수집되기 때문이고, 고도와 기울기는 DCC 장비의 센서에서 측정된 값과 GPS 수신기를 통해 계산되는 값이 저장되기 때문이다.

원시 데이터를 정밀 분석하기 위해서는 입력데이터를 분석하기 용이한 데이터로 처리하는 정제 메카니즘이 필요하다. 이는 측정 센서의 오류 및 주행 중 정차 등 특이사항이 그대로 저장되기 때문인데, <그림8>의 데이터를 살펴보면 GPS 좌표값이 중복돼있는 것을 확인할 수 있다.

	경도	위도	셀리지온경도	셀리지온위도	GPS고도	속도	온도	습도	압력	고도	배터리	시간	기울기1	기울기2
2	126.5504	33.4549	126.5507745	33.4567423	-210	2	11.5	55.1	997.999	703.12	64.5	2014-01-18 오후 8:44:16	0	0
3	126.5504	33.4549	126.5507745	33.4567423	-210	2	11.5	55.1	997.999	703.13	64.5	2014-01-18 오후 8:44:17	-0.1	0
4	126.5504	33.4549	126.5507745	33.4567423	-209.8	2	11.5	55.1	998.047	702.73	64.5	2014-01-18 오후 8:44:18	-0.2	0
5	126.5504	33.4549	126.5507745	33.4567423	-209.7	2	11.5	55.1	998.018	703	64.5	2014-01-18 오후 8:44:19	-0.2	0
6	126.5504	33.4549	126.5507745	33.4567423	-209.6	1	11.5	55.1	998.042	702.79	64.5	2014-01-18 오후 8:44:20	-0.2	0
7	126.5504	33.4549	126.5507745	33.4567423	-209.6	2	11.5	55.1	998.002	703.15	64.5	2014-01-18 오후 8:44:21	0.6	0
8	126.5504	33.4549	126.5507745	33.4567423	-209.6	2	11.5	55.1	998.012	703.07	64.5	2014-01-18 오후 8:44:22	9.6	5.2
9	126.5504	33.45489	126.5507745	33.4567423	-209.5	2	11.5	55.1	998.027	702.95	64.5	2014-01-18 오후 8:44:23	9.6	0
10	126.5504	33.45488	126.5507745	33.4567423	-209.5	5	11.5	55.1	998.012	703.1	64.5	2014-01-18 오후 8:44:24	10.3	0
11	126.5504	33.45487	126.5507771	33.4567448	-209.5	7	11.5	55.1	997.939	703.73	64.5	2014-01-18 오후 8:44:25	10.3	0
12	126.5504	33.45485	126.5507988	33.4567211	-209.5	10	11.4	55.1	997.901	704.04	64.5	2014-01-18 오후 8:44:26	4.6	0
13	126.5504	33.45485	126.55083	33.4566828	-209.5	10	11.4	55.1	997.884	704.16	64.5	2014-01-18 오후 8:44:27	4.6	0.4
14	126.5505	33.45476	126.5508611	33.4566286	-209.4	20	11.5	55.1	997.869	704.3	64.5	2014-01-18 오후 8:44:28	-11.3	-0.9
15	126.5505	33.4547	126.5508858	33.4565635	-209.5	22	11.5	55.1	997.908	704	64.5	2014-01-18 오후 8:44:29	-11.3	0
16	126.5505	33.45465	126.5509071	33.4564985	-209.5	21	11.5	55.1	997.896	704.1	64.5	2014-01-18 오후 8:44:30	-11.5	0
17	126.5505	33.45461	126.5509215	33.4564443	-209.5	18	11.5	55.1	997.924	703.88	64.5	2014-01-18 오후 8:44:31	-11.5	0
18	126.5505	33.45461	126.5509316	33.4564033	-209.5	18	11.5	55.1	997.918	703.94	64.5	2014-01-18 오후 8:44:32	-2.3	0
19	126.5505	33.45455	126.5509385	33.4563758	-209.5	12	11.5	55.1	997.894	704.16	64.5	2014-01-18 오후 8:44:33	-2.3	0
20	126.5505	33.45454	126.5509436	33.4563581	-209.5	10	11.5	55.5	997.901	704.1	64.5	2014-01-18 오후 8:44:34	-5.7	-2.6
21	126.5505	33.45452	126.5509468	33.456346	-209.6	7	11.5	55.5	997.867	704.39	64.5	2014-01-18 오후 8:44:35	-5.7	-2.6
22	126.5505	33.45452	126.5509494	33.4563389	-209.6	5	11.5	55.5	997.92	703.96	64.5	2014-01-18 오후 8:44:36	0.2	-2.6
23	126.5505	33.45452	126.5509503	33.4563373	-209.6	5	11.5	55.5	997.885	704.27	64.5	2014-01-18 오후 8:44:37	0.2	-5.2
24	126.5505	33.45451	126.5509509	33.4563368	-209.7	3	11.5	55.5	997.914	704.05	64.5	2014-01-18 오후 8:44:38	0	-5.2
25	126.5505	33.45451	126.5509509	33.4563368	-209.8	2	11.5	55.5	997.9	704.16	64.5	2014-01-18 오후 8:44:39	0	-5.2
26	126.5505	33.45451	126.5509509	33.4563368	-209.8	0	11.5	55.1	997.901	704.16	64.5	2014-01-18 오후 8:44:40	-0.2	0
27	126.5505	33.45452	126.5509509	33.4563368	-209.8	0	11.5	55.1	997.879	704.33	64.5	2014-01-18 오후 8:44:41	-0.2	0
28	126.5505	33.45452	126.5509509	33.4563368	-209.8	0	11.5	55.1	997.869	704.44	64.5	2014-01-18 오후 8:44:42	0.2	-5.2
29	126.5505	33.45451	126.5509509	33.4563368	-209.9	0	11.5	55.1	997.883	704.32	64.5	2014-01-18 오후 8:44:43	0.2	-5.2
30	126.5505	33.45451	126.5509509	33.4563368	-210	0	11.5	55.1	997.863	704.5	64.5	2014-01-18 오후 8:44:44	0	-5.2

그림 8. 원시 데이터의 문제점

또한, 통계적 분석 및 다른 알고리즘에 적용을 쉽게 하기 위해서는 데이터의 정규화 과정도 필요하다. 특히 전기자동차 트립마다 초기 잔량이 다른 경우, 이를 균일화하여야 한다.

본 논문에서는 이러한 원시 데이터의 문제점을 해결하는 정제 메카니즘을 빅데이터 처리기술인 Hadoop을 사용하여 제시하였다. Linux를 탑재한 PC에 원시 데이터를 저장하고 일부 필드를 Hadoop이 읽을 수 있는 형태로 변환한다. 그리고 주행거리에 따른 배터리 소모량을 분석하기 위해 GPS 좌표의 중복을 제거하며 통계적 분석이 용이하게 SoC에 대한 정규화 작업을 진행하였다.

2. Hadoop Pig 인터페이스

'광역경제권연계협력사업'의 총괄주관기관인 제주대학교 전기차사업단의 웹서버인 jejue-vrc.jejunu.ac.kr에는 웹 컨텐츠 뿐 만 아니라 전기자동차의 주행 패턴을 분석하기 위한 데이터와 빅데이터 처리기술인 Hadoop과 Hadoop의 서브프로젝트인 Hadoo

p Pig(ve-r 0.12.1)가 설치되어있다.

Hadoop Pig는 Hadoop 환경 하에서 실행되는 데이터 플로우 언어이며, 다양한 데이터 타입과 대용량의 데이터를 고속처리 할 수 있다. 또한, 복잡한 Map-Reduce를 대체하는 Pig Latin을 제공한다. 따라서 고도의 병렬/분산 처리를 필요로 하지 않는 경우에 적합한 분석 환경으로 알려져있다.

전기차사업단 웹서버에 설치된 Hadoop과 Hadoop Pig의 버전은 <그림9>와 같다.

```
kkwnana@evrc-SERVER:~$ hadoop version
Hadoop 1.2.1
Subversion https://svn.apache.org/repos/asf/hadoop/common/branches/branch-1.2 -r 1503152
Compiled by mattf on Mon Jul 22 15:23:09 PDT 2013
From source with checksum 6923c86528809c4e7e6f493b6b413a9a
This command was run using /home/jhlee/hadoop-1.2.1/hadoop-core-1.2.1.jar
kkwnana@evrc-SERVER:~$ pig version
2015-05-28 15:23:15,060 [main] INFO  org.apache.pig.Main - Apache Pig version 0.12.1 (r1585011) compiled Apr 05 2014, 01:41:34
2015-05-28 15:23:15,060 [main] INFO  org.apache.pig.Main - Logging error messages to: /home/kkwnana/pig_1432794195059.log
2015-05-28 15:23:15,201 [main] ERROR org.apache.pig.Main - ERROR 2997: Encountered IOException. File version does not exist.
Details at logfile: /home/kkwnana/pig_1432794195059.log
```

그림 9. Hadoop과 Hadoop Pig의 버전 정보

Hadoop Pig를 실행하기 위해서는 환경변수의 설정을 해야 하며 <그림10>에서 보는 바와 같이 .profile에 vi 에디터를 이용하여 수정한다.

```
1 EVRC
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH=".:HOME/bin:$PATH"
fi
export JAVA_HOME=/usr/lib/jvm/jdk1.6.0_38
export HADOOP_INSTALL=/home/jhlee/hadoop-1.2.1
export PATH=$PATH:$HADOOP_INSTALL/bin:$JAVA_HOME/bin
export PIG_HOME=/home/jhlee/pig-0.12.1
export PIG_HISPSIZE=512
export PIG_CLASSPATH=$HADOOP_INSTALL/conf
export PATH=$PATH:$PIG_HOME/bin
```

그림 10. Hadoop Pig의 환경변수 설정

Hadoop Pig는 <그림11>과 같이 Grunt 쉘에서 Pig Latin을 입력하여 대화식으로 실행하는 방법과 <그림12>와 같이 배치파일을 만들어 쉘에서 실행하는 두 가지 방식을 지원한다. Hadoop의 실행모드는 Map-Reduce모드(Hadoop 모드)와 로컬모드 두 가지가 있는데, 기본적으로 Map-Reduce모드를 사용한다.

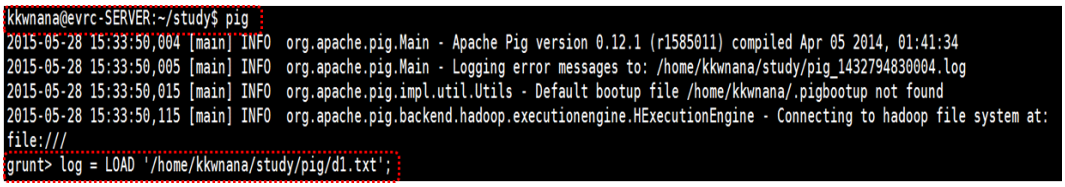


그림 11. Grunt 쉘을 이용한 대화식 실행 방법

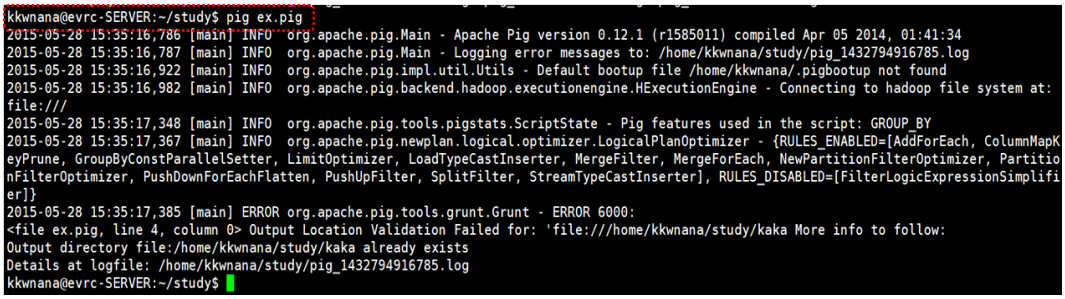


그림 12. 배치파일(ex.pig)를 이용한 쉘에서의 실행 방법

3. 정제 메카니즘

본 논문에서는 전기자동차 주행의 원시 데이터를 누적 거리별 배터리 소모량에 대한 데이터로 가공하는 메카니즘을 제안한다. 제안하는 메카니즘은 GPS 좌표의 중복값 제거와 SoC에 대한 정규화, GPS 좌표를 통해 이동 거리 계산을 수행한다.

3.1 Pig Latin

Pig Latin을 사용하여 원시 데이터를 입력받아 GPS의 중복 좌표의 제거와 배터리의 잔량을 정규화 시키는 Pig Latin은 <그림13>과 같으며, 배치파일을 만들어 Pig Latin을 실행시켰다.

```
log = LOAD '/home/kkwnana/study/pig/d1.txt' USING PigStorage('\t') AS (f1:
    double, f2: double, f3: double, f4: double, f5: chararray, f6: chararray, f7:
    chararray, f8: chararray, f9: chararray, f10: chararray, f11: double, f12:
    datetime, f13: double, f14: double);
temp = GROUP log BY (f1, f2);
temp1 = FOREACH temp  GENERATE MAX($1.f12), $0.f1, $0.f2, MAX($1.f11);
proj1 = DISTINCT temp1;
temp2 = GROUP proj1 ALL;
d = FOREACH temp2 GENERATE MAX(proj1.$3) as val1, MIN(proj1.$3) as val2;
proj2 = FOREACH proj1 GENERATE $1, $2, ($3-d.val2)/(d.val1-d.val2);
STORE proj2 into 'evsoc';
```

그림 13. SoC 정규화를 위한 Pig Latin

Pig Latin을 동작을 살펴보면 :

i. log alias에 원시 데이터를 저장시키며, 각 필드의 데이터 타입을 정의한다.

ii. log alias의 중복된 GPS 좌표값을 제거하기 위해 f1, f2 필드를 그룹화시켜 temp alias를 생성한다.

-> 원시 데이터 중 GPS 좌표는 f1, f2, f3, f4 필드에 존재한다. 이 2개의 GPS 좌표는 WGS84좌표계(f1, f2)와 Bessel좌표계(f3, f4)를 사용한다. 본 논문에서는 WGS84 좌표계를 사용하는 f1, f2를 사용하였다.

-> Pig Latin에서의 중복값 제거는 DISTINCT 명령어를 사용하나, alias 전체 필드에 적용되기 때문에 본 논문에서는 그룹화를 통하여 중복값을 제거하였다.

iii. temp alias의 시간, GPS 좌표, SoC 필드만을 사용하여 temp1 alias를 생성한 후 시간에 대해 정렬을 실시해 proj2 alias를 생성한다.

-> Pig Latin에서 그룹화를 실행하면 그룹화된 필드를 기준으로 정렬이 되기 때문에 temp alias은 GPS 좌표값으로 정렬이 된다. 그러나 GPS 좌표는 시간순서로 정렬이 되어야 이동구간을 파악할 수 있기 때문에 새로운 alias를 생성해야한다.

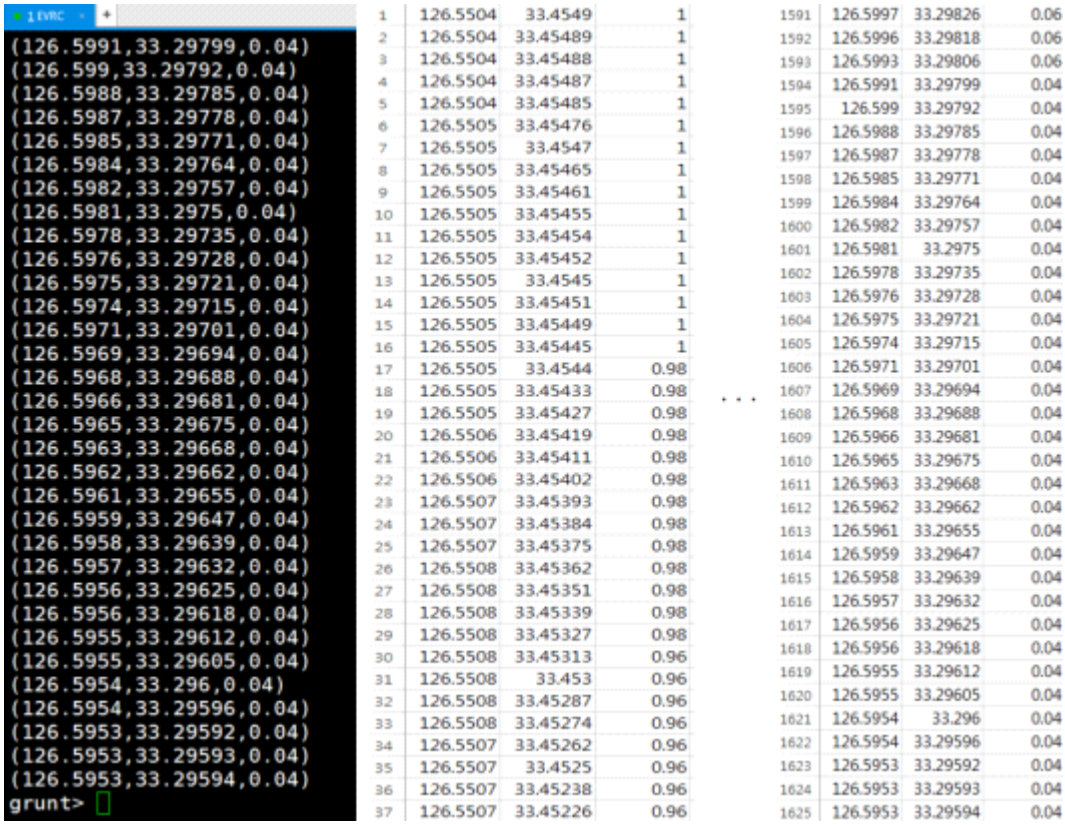
iv. d alias에 SoC의 최대값과 최소값을 저장한다.

v. proj1 alias의 GPS 좌표와 SoC를 정규화시켜 proj2 alias를 생성한다.

vi. 결과값을 'evsoc' 폴더에 저장한다.

이 Pig Latin을 수행한 결과 약 300여 개의 중복된 GPS 좌표가 삭제되었으며, SoC

가 0~1 사이의 값으로 변환된 것을 확인할 수 있다. 결과는 <그림14>와 같다.



```
(126.5991,33.29799,0.04)
(126.599,33.29792,0.04)
(126.5988,33.29785,0.04)
(126.5987,33.29778,0.04)
(126.5985,33.29771,0.04)
(126.5984,33.29764,0.04)
(126.5982,33.29757,0.04)
(126.5981,33.2975,0.04)
(126.5978,33.29735,0.04)
(126.5976,33.29728,0.04)
(126.5975,33.29721,0.04)
(126.5974,33.29715,0.04)
(126.5971,33.29701,0.04)
(126.5969,33.29694,0.04)
(126.5968,33.29688,0.04)
(126.5966,33.29681,0.04)
(126.5965,33.29675,0.04)
(126.5963,33.29668,0.04)
(126.5962,33.29662,0.04)
(126.5961,33.29655,0.04)
(126.5959,33.29647,0.04)
(126.5958,33.29639,0.04)
(126.5957,33.29632,0.04)
(126.5956,33.29625,0.04)
(126.5956,33.29618,0.04)
(126.5955,33.29612,0.04)
(126.5955,33.29605,0.04)
(126.5954,33.296,0.04)
(126.5954,33.29596,0.04)
(126.5953,33.29592,0.04)
(126.5953,33.29593,0.04)
(126.5953,33.29594,0.04)
grunt>
```

1	126.5504	33.4549	1
2	126.5504	33.45489	1
3	126.5504	33.45488	1
4	126.5504	33.45487	1
5	126.5504	33.45485	1
6	126.5505	33.45476	1
7	126.5505	33.4547	1
8	126.5505	33.45465	1
9	126.5505	33.45461	1
10	126.5505	33.45455	1
11	126.5505	33.45454	1
12	126.5505	33.45452	1
13	126.5505	33.4545	1
14	126.5505	33.45451	1
15	126.5505	33.45449	1
16	126.5505	33.45445	1
17	126.5505	33.4544	0.98
18	126.5505	33.45433	0.98
19	126.5505	33.45427	0.98
20	126.5506	33.45419	0.98
21	126.5506	33.45411	0.98
22	126.5506	33.45402	0.98
23	126.5507	33.45393	0.98
24	126.5507	33.45384	0.98
25	126.5507	33.45375	0.98
26	126.5508	33.45362	0.98
27	126.5508	33.45351	0.98
28	126.5508	33.45339	0.98
29	126.5508	33.45327	0.98
30	126.5508	33.45313	0.96
31	126.5508	33.453	0.96
32	126.5508	33.45287	0.96
33	126.5508	33.45274	0.96
34	126.5507	33.45262	0.96
35	126.5507	33.4525	0.96
36	126.5507	33.45238	0.96
37	126.5507	33.45226	0.96

...

1591	126.5997	33.29826	0.06
1592	126.5996	33.29818	0.06
1593	126.5993	33.29806	0.06
1594	126.5991	33.29799	0.04
1595	126.599	33.29792	0.04
1596	126.5988	33.29785	0.04
1597	126.5987	33.29778	0.04
1598	126.5985	33.29771	0.04
1599	126.5984	33.29764	0.04
1600	126.5982	33.29757	0.04
1601	126.5981	33.2975	0.04
1602	126.5978	33.29735	0.04
1603	126.5976	33.29728	0.04
1604	126.5975	33.29721	0.04
1605	126.5974	33.29715	0.04
1606	126.5971	33.29701	0.04
1607	126.5969	33.29694	0.04
1608	126.5968	33.29688	0.04
1609	126.5966	33.29681	0.04
1610	126.5965	33.29675	0.04
1611	126.5963	33.29668	0.04
1612	126.5962	33.29662	0.04
1613	126.5961	33.29655	0.04
1614	126.5959	33.29647	0.04
1615	126.5958	33.29639	0.04
1616	126.5957	33.29632	0.04
1617	126.5956	33.29625	0.04
1618	126.5956	33.29618	0.04
1619	126.5955	33.29612	0.04
1620	126.5955	33.29605	0.04
1621	126.5954	33.296	0.04
1622	126.5954	33.29596	0.04
1623	126.5953	33.29592	0.04
1624	126.5953	33.29593	0.04
1625	126.5953	33.29594	0.04

그림 14. Pig Latin으로 정제된 데이터

결과를 살펴보면 최종 레코드의 SoC가 0이 아닌 0.04의 값을 가지게 된다. 이유는 전기자동차는 내리막 또는 브레이크 제동 시 회생 제동력을 통하여 배터리를 충전시키기 때문이며, 실제 주행한 516도로는 한라산을 횡단하는 도로이기 때문에 도착지점은 내리막길이다.

Pig Latin을 사용하면 간단한 코딩으로 데이터의 가공이 가능하였다. 그러나 Pig Latin에서 지원되는 명령어로는 GPS 좌표값을 통해 실제 이동거리를 계산하기에는 다소 무리가 있다. 이는 Pig UDF(User Define Function)를 통하여 사용자 정의 함수를 만들어 적용하여야 한다.

3.2 Pig UDF

Hadoop Pig는 사용자 정의함수인 Pig UDF를 제공한다. Pig UDF는 자바로 구현되며 Hadoop Pig와 자바의 인터페이스가 중요하며, 사용자가 원하는 기능을 쉽게 추가할 수 있다. UDF의 생성과정은 먼저 자바프로그램 작성 후 컴파일과정을 거친 후 JAR 파일의 생성, Pig Latin 작성 후 자바에 의해 프로그램이 수행되는 과정을 거친다. 3.1의 Pig Latin으로는 GPS 좌표의 중복값 제거와 SoC의 데이터 정규화는 쉽게 되었으나 Pig Latin에서는 GPS 좌표 변환에 필요한 수학연산에 대한 명령어를 제공하지 않기 때문에 본 절에서 Pig UDF를 이용하여 데이터를 가공하였다.
3.1의 결과를 입력데이터로 하여 Pig UDF를 사용하여 GPS 좌표로 이동 거리를 계산하였다.
JAR 파일을 생성하기 위한 자바 코드는 <그림15>와 같다.

```
package mybag;
import java.io.IOException;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Iterator;
import java.util.Date;

import org.apache.pig.EvalFunc;
import org.apache.pig.PigWarning;
import org.apache.pig.data.Tuple;
import org.apache.pig.data.DataBag;
import org.apache.pig.data.BagFactory;
import org.apache.pig.data.TupleFactory;
import org.apache.pig.impl.util.WrappedIOException;


public class mybag extends EvalFunc <DataBag> {
        public DataBag exec (Tuple input) throws IOException {

BagFactory bagFactory = BagFactory.getInstance();
TupleFactory tupleFactory = TupleFactory.getInstance();
                DataBag bag = (DataBag) input.get(0);
                DataBag ret = bagFactory.newDefaultBag();
                Iterator it = bag.iterator();
```

```
                Double d = 0.0, here  =0.0, dx, dy;
                Double x, y, px, py, s;
                Tuple t;

                dx = 31.0 /0.0002777778;
                dy = 25.0 /0.0002777778;
                dx = dx*dx;
                dy = dy*dy;

                if (! it.hasNext())
                        return null;
                t = (Tuple) it.next();
                px = ((Double) t.get(0));
                py = ((Double) t.get(1));

                while (it.hasNext()) {
                        t = (Tuple) it.next();
                        try {
                                x = ((Double) t.get(0));
                                y = ((Double) t.get(1));
                                if ((x.equals(px)) && (y.equals(py)))
                                        continue;
                                here = here + Math.sqrt(
                                        (px-x)*(px-x)* dx +
                                        (py-y)*(py-y)* dy);
                                px = x; py = y;
                                s = ((Double) t.get(3));
                                Tuple m = tupleFactory.newTuple(2);
                                m.set(0, here);
                                m.set(1, s);
                                ret.add(m);

                        } catch(Exception e) {
                                return null;
                        }
                }
                return ret;
        }
}
```

그림 15. GPS 좌표를 통한 거리계산 코드

i. 먼저 Hadoop Pig를 사용하기 위해 Hadoop에서 지원하는 클래스를 import

를 사용하여 지정해준다.

ii. DataBag 타입의 입력에서 GPS 좌표를 통해 거리를 계산하는 코드를 적용한다.

-> 변수 x, y, px, py, dx, dy를 사용하여 이동 거리를 계산하여 here 변수에 저장을 하였으며, 이동 거리와 SoC에 대한 정규화 값을 DataBag 타입으로 반환한다.

Pig UDF를 실행하기 위한 Pig Latin은 <그림16> 같다

```
REGISTER mybag.jar;
log = LOAD '/home/kkwnana/study/final/evsoc/part-m-00000' USING PigStorage('\t
        ') AS (f1: double, f2: double, f3: double);
temp4 = GROUP log ALL;
result = FOREACH temp4 GENERATE mybag.mybag(log);
STORE result into 'result';
```

그림 16. Pig UDF를 위한 Pig Latin

i. Pig UDF를 실행하기 위한 JAR파일을 레지스터 한다.

ii. 3-1의 결과값으로 log alias를 생성한다.

iii. log alias의 값을 Pig UDF에 입력값으로 하여 실행시킨다.

Pig UDF의 실행은 <그림17>과 같이 쉘에서 실행한다.

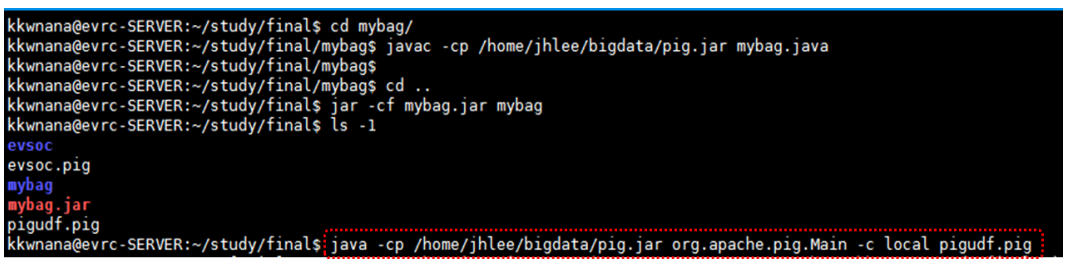


그림 17. Pig UDF의 실행방법

Pig UDF의 결과물은 result폴더에 생성되며 그 값은 <그림 18>과 같다.

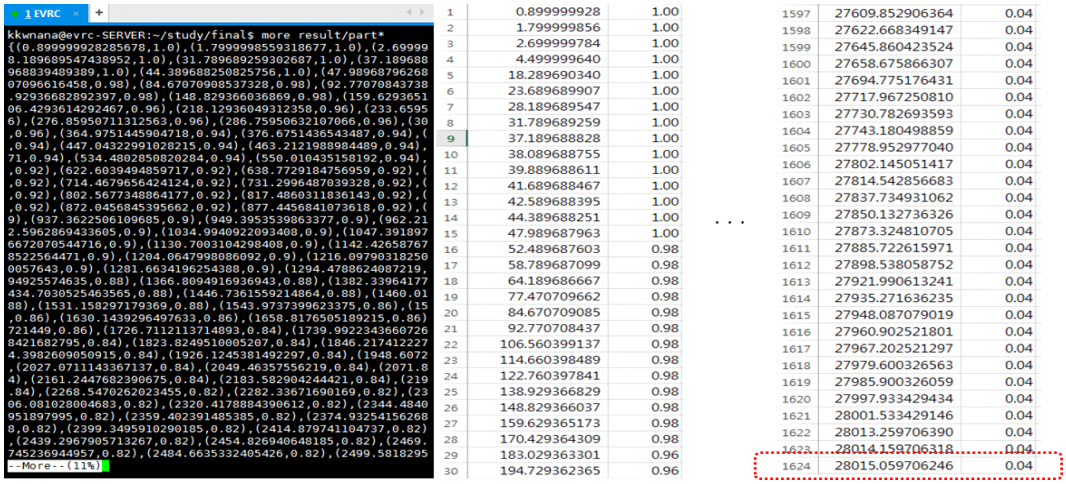


그림 18. Pig UDF를 통하여 정제된 데이터

지도 서비스(네이버 지도)를 이용하여 출발지점과 도착지점의 거리를 계산 하였을 때는 약 26Km의 거리가 계산되었으나 본 메카니즘을 사용하였을 때는 약 28km로 계산되었다. 이는 전기자동차에 장착된 GPS 좌표의 수집장치의 오차와 GPS 좌표로 이동 거리를 계산하는 수학적 방법이 다르기 때문으로 판단된다.

IV. 결과

본 장에서는 제안 메카니즘에 대한 결과를 분석한다. Hadoop Pig의 DataBag 타입으로 출력된 SoC의 소모 특성을 그래프로 변환하여 SoC의 변화에 대해 알아보며, 그리고 동일구간에 대한 여러 데이터의 비교를 통해 운전자의 운전 습관, 주변 환경 요인 등의 영향에 의해 배터리 소모량이 달라지는지를 확인한다.

먼저 <그림19>에서 보는 바와 같이 516도로를 통해 제주시와 서귀포시를 왕복으로 운행하는 차량에서 추출된 스트림을 대상으로 하였다. B지점은 이동 경로 상 최대 고도인 지점이다. A지점에서 B지점까지의 거리는 약 12.5Km이며 전반적으로 오르막구간이며, B지점에서 C지점까지의 거리는 약 13.5km이며 전반적으로 내리막구간이다.

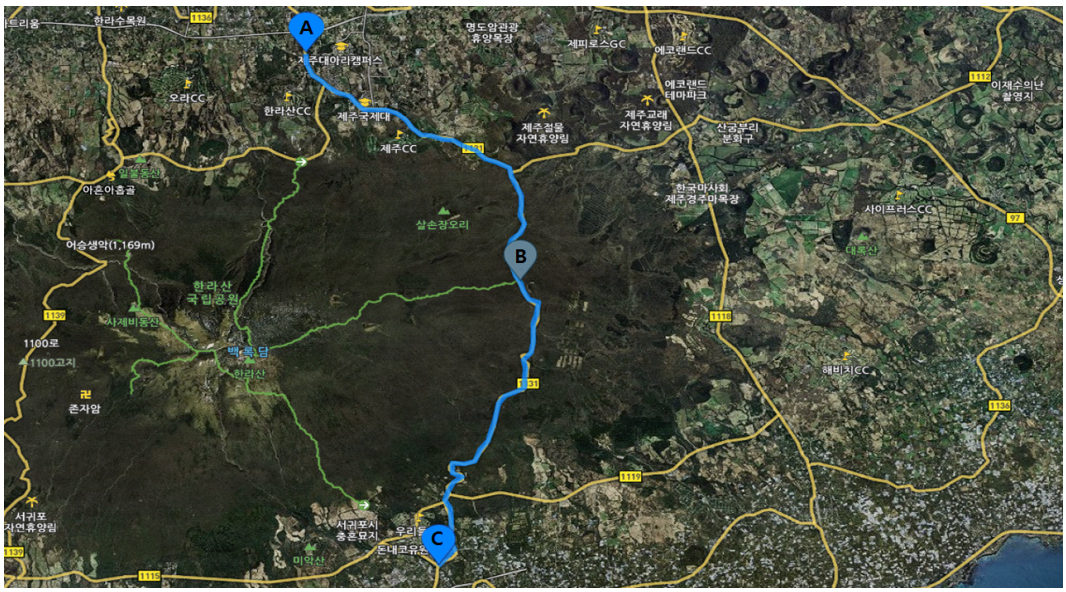


그림 19. 이동구간

정방향(A지점에서 C지점)으로 운행한 차량에서 추출된 스트림을 본 논문에서 제안된 메카니즘으로 정제된 데이터를 도식화하면 <그림20>과 같다. 여기서 x축은 시작점으로부터의 이동거리를 나타내며 GPS 좌표들로부터 변환된 결과이다. y축은 SoC를 나타내는데 이동 중 최소 SoC와 최대 SoC를 각각 0.0과 1.0으로 정규화하였다.

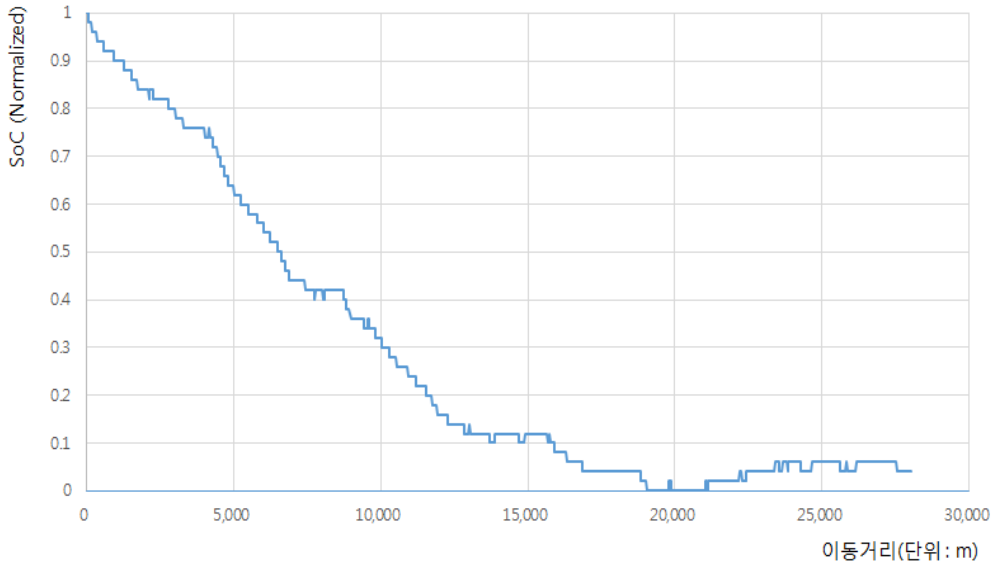


그림 20. 이동거리별 SoC 변화량

<그림20>을 살펴보면 오르막구간에서는 지속해서 배터리가 소모되는 것을 확인할 수 있으며 일부 구간에 대해 회생 제동을 통하여 배터리 소모가 없는 구간과 충전되는 구간이 있음을 확인할 수 있다. 또한, <그림21>와 <그림22>에서 보는 바와 같이 동일 구간에 대한 중복 측정을 통해 여러 데이터를 비교해보면 측정할 때마다 배터리 소모량이 달라지는 것을 확인할 수 있다.

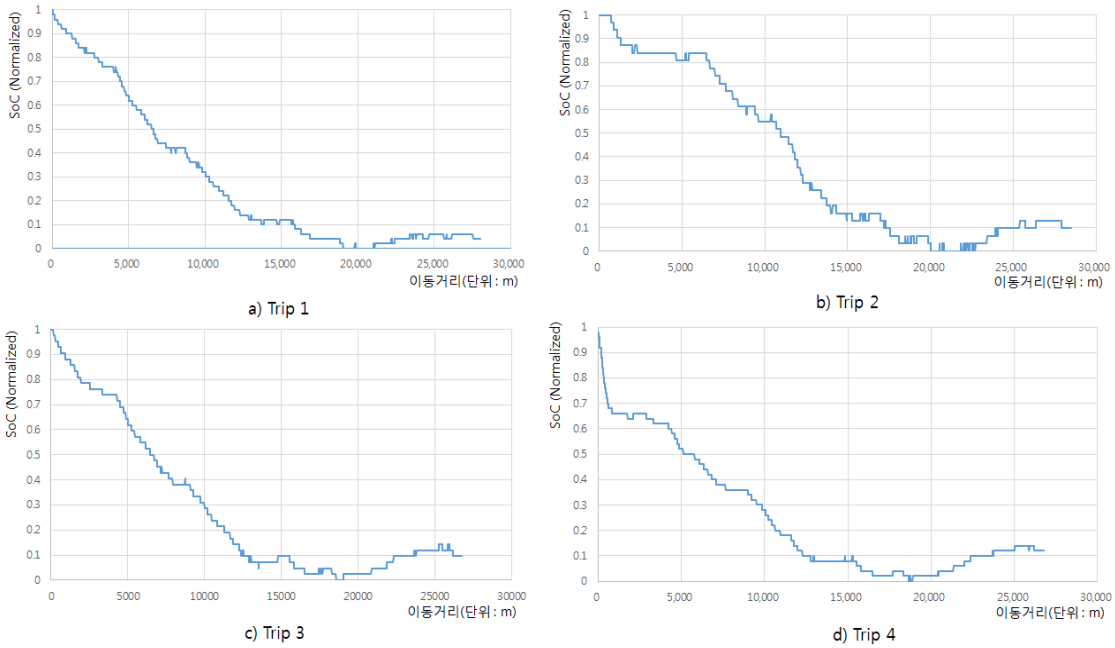


그림 21. 정방향 이동구간(A->C) 대한 SoC 변화량(1)

<그림21>는 특히 주행 초기에 급가속이나 급제동을 한 경우 SoC 패턴의 차이가 나타나며, 보다 많은 수의 SoC에 대한 스트림을 수집하면 오차를 마스킹하고 공통적인 소모 모델을 구할 수 있을 것으로 판단된다. a)의 그래프는 <그림19>와 같은 그래프이다. b)의 그래프는 a)에 비해 배터리 소모가 완만하게 이루었다는 것을 확인할 수 있으며, 이는 운전자가 경제운전 및 회생제동에 신경을 쓰며 운전하였다고 예상할 수 있다. c)의 그래프는 a)의 그래프와 유사하다는 것을 확인할 수 있다. d)의 그래프를 살펴보면 a)의 그래프보다 배터리 소모가 급격하게 이루어졌다는 것을 알 수 있었다. 그래프마다 SoC의 변화량이 다른 이유는 시작점에서 급가속 혹은 다른 차량의 간섭 등의 원인으로 배터리 소모가 많이 되었다고 예상할 수 있으며, 운전자의 운전습관과 주변 환경요인에 따라 달라졌다는 것을 예상할 수 있다. <그림22>을 통해 쉽게 비교할 수 있다.

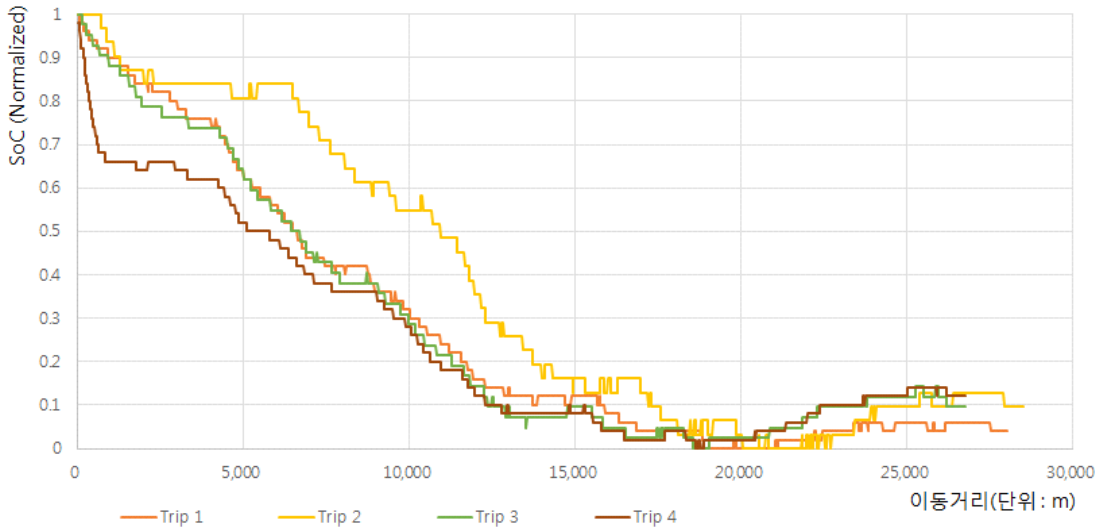


그림 22. 정방향 이동구간(A->C)에 대한 SoC 변화량(2)

역방향으로 주행(C지점에서 A지점으로)한 차량에 대한 배터리 변화량은 <그림23>과 <그림24>를 통해 확인할 수 있다. SoC에 대한 변화는 정방향의 변화량과 유사하지만, 오르막구간이 정방향 운행과 비교하면 약 1km 더 길기 때문에 배터리가 지속적으로 소모되는 구간이 정방향 이동구간보다 긴 것을 확인할 수 있다.

또한, A지점에서 B지점까지의 배터리 소모량의 누적 합과 B지점에서 A지점까지의 배터리 소모량의 누적합의 평균값(0.158 : 0.061)을 비교하면 내리막구간에서는 회생제동에 의해 배터리 소모가 오르막구간을 주행할 때보다 약 61% 정도 적게 소모되는 것을 알 수 있었다. 그리고 동일지점(정방향 이동 시 약 4.7km 지점)에서 오르막과 내리막 구간의 SoC 변화를 살펴보면 0.2 : 0.227로 회생제동으로 충전되는 양이 많게 나타났다. 이는 A지점에서 C지점을 왕복 주행 하더라도 SoC의 소비량은 다르지만, 정규화를 통해 0.0 ~ 1.0 사이로 변환하였기에 생기는 오차로 예상된다. 그리고 SoC 측정 장치에서 수집되는 데이터도 소수점 첫째자리만 측정이 가능하기에 세밀한 분석은 어려울 것으로 예상하며, 표본의 수가 적어 통계적으로 검증은 하지 못하였다.

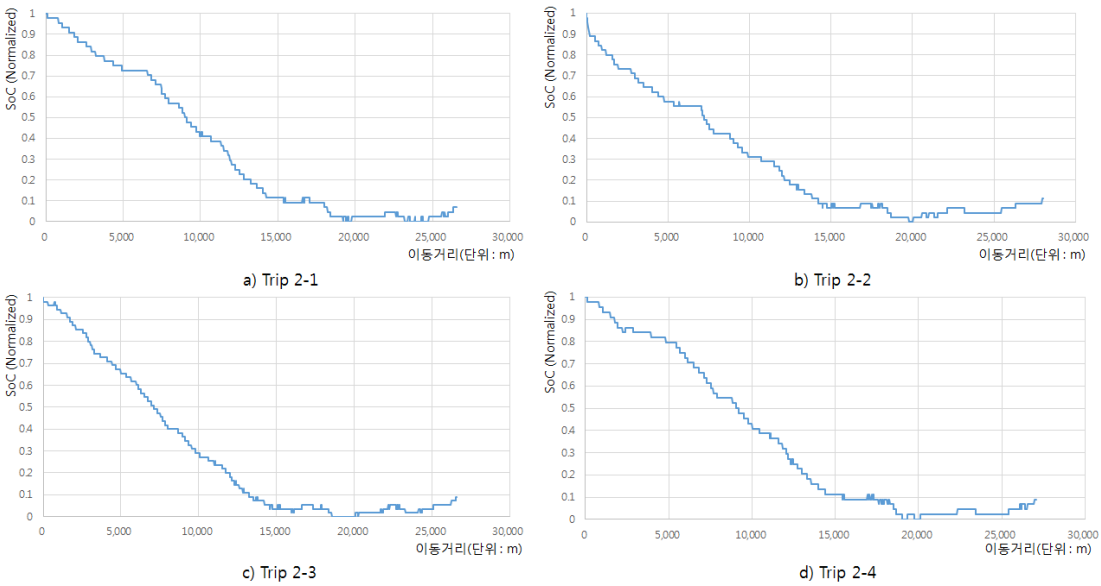


그림 23. 역방향 이동구간(C->A)에 대한 SoC 변화량(1)

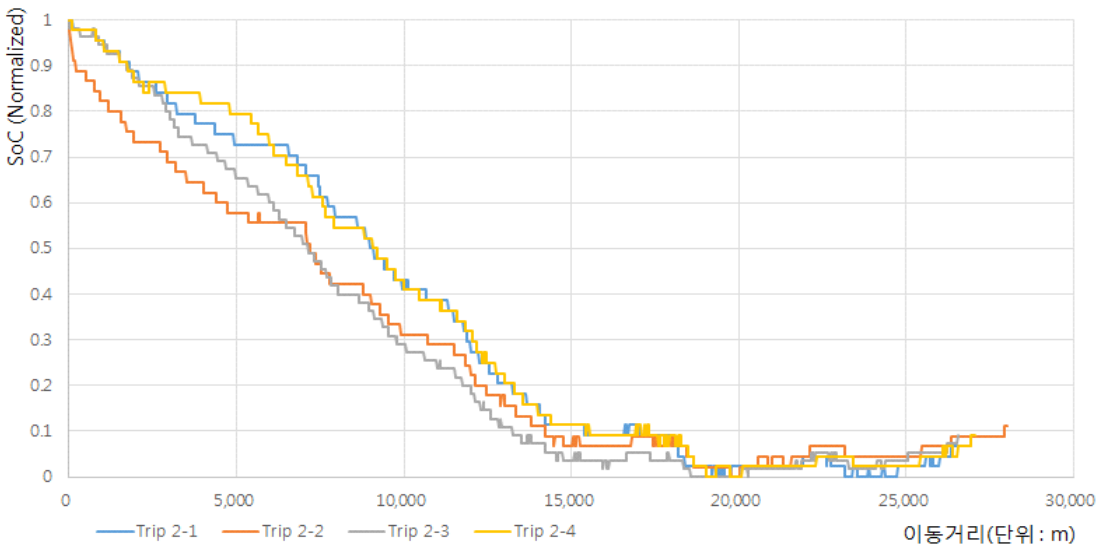


그림 24. 역방향 이동구간(C->A)에 대한 SoC 변화량(2)

그리고 제안 메카니즘을 활용하면 주요 도로에 대한 SoC 소비 모델을 만드는 데 도움을 줄 수 있을 것이다. 주요 도로에 대한 SoC 소비 모델을 활용하면 <그림25>에서 보는 바와 같이 길 안내 서비스를 할 때 DB 조회와 약간의 on-line 계산만으로 경로 탐색과 SoC 소비량을 예상하는 데 활용할 수 있을 것이다.

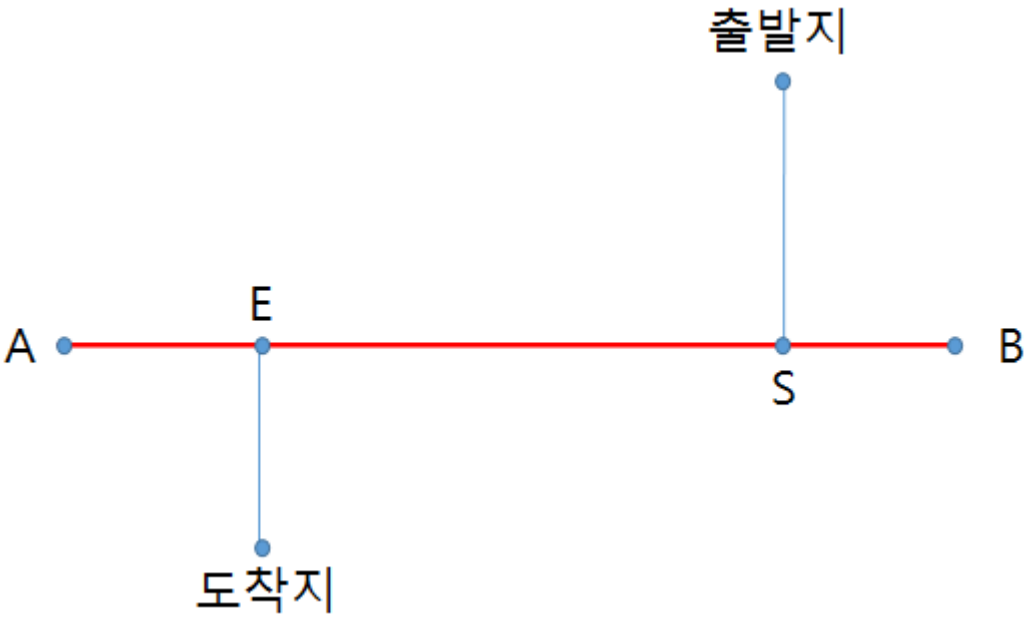


그림 25. 모델링 된 도로를 이용하는 길안내 서비스의 예

i. A -> B : 모델링 된 도로.
ii. 출발지 -> S : 거리기반으로 최단거리 및 SoC 소비량 계산.
iii. S -> E : DB에 저장된 모델링된 데이터를 사용.
iv. E -> 도착지 : 거리기반으로 최단거리 및 SoC 소비량 계산.

V. 결론

본 논문에서는 스마트그리드와 빅데이터 처리기술을 접목해 전기자동차 주행 데이터를 정제하는 메카니즘을 설계하고 구현하였다. '광역경제권연계협력사업'을 수행하였던 제주대학교 전기차사업단의 도움을 받아 전기자동차의 배터리 소모량에 대한 원시 데이터를 사용하여 Linux상에서 Hadoop Pig를 사용하여 분석하기 쉬운 데이터로 정제하는 스크립트를 작성하였으며, 제안된 메카니즘으로 정제된 데이터를 그래프로 표현하고, 이를 통해 배터리 소모량의 변화 정도를 살펴보았다.

Hadoop Pig는 고도의 병렬/분산 처리를 필요로 하지 않는 경우에 적합한 분석 환경으로 알려져 있으며, Pig UDF라는 사용자 정의 함수를 사용하여 여러 기능을 손쉽게 추가할 수 있다. 본 논문에서는 Pig Latin을 사용하여 14개의 필드값을 가지는 원시 데이터에 대하여 중복된 GPS 좌표값 제거 및 SoC를 정규화시켰으며, Pig UDF를 사용하여 GPS 좌표값으로부터 이동 거리를 계산하여 최종적으로 누적 이동거리와 정규화된 SoC로 데이터를 정제하였다.

그리고 동일구간에 대한 여러 데이터를 비교하여 운전자의 운전 습관 및 주변 환경 요인으로 인해 배터리 소모율이 달라지는 것을 확인하였으며, 역방향에 대한 데이터를 같이 살펴봄으로써 회생 제동이 SoC 변화에 미치는 영향을 확인하였다.

제안 메카니즘을 활용하면 통계적 분석 및 예측 알고리즘에 적용하기 쉬운 정제 데이터가 생성된다. 그리고 지속적인 연구개발을 통해 측정 센서의 정밀도를 높이고, 전기자동차의 보급이 활성화됨에 따라 방대한 양의 주행데이터가 생성되면, 이 방대한 데이터의 분석을 위한 전처리 작업을 수행할 수 있을 것이다. 이는 도로의 경사도에 대한 가중치 계산, 온도 및 습도 변화에 따른 배터리 소모량의 변화 예측, 운전 습관에 따른 배터리 소모량의 변화 등을 분석하는 데 도움을 줄 수 있으며, 전기

자동차 배터리의 효율적 사용을 가능하게 할 수 있을 것으로 기대된다.
더욱이, 제주도는 전기자동차와 충전설비 등의 보급이 세계적으로도 우수한 지역으로 이들 스마트그리드 엔티티들에서 실시간 모니터링되는 데이터 스트림들은 전기자동차와 관련된 다양한 비즈니스 모델을 창출할 것이다. 본 논문에서 제안된 전기자동차 데이터의 처리방식은 전기자동차에 특화된 신규 서비스개발은 물론 다른 스트림(예를 들면, 충전기 모니터링 데이터, 빌딩 등에서의 전력소모 데이터 등)과 결합하여 거대한 도시레벨의 빅데이터 플랫폼의 빌딩 블록으로서의 역할을 담당하게 될 것이다.

VI. 참고문헌

[1] 지능형전력망협회,(주)한국전력신문사, 2013.12, 2014 스마트그리드 연감

[2] 손상훈, 전기자동차 이용 행태 및 효과 분석, 제주발전연구원

[3] 제주대학교 전기차사업단, online, http://jejuevrc.jejunu.ac.kr/

[4] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler 2010. The Hadoop Distributed File System, Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on , pp.1-10

[5] Jin-Woo Lee and Su-Kyoung Kim, Complementary research and Analysis for hadoop, 한국컴퓨터정보학회 하계학술대회 논문집 제20권 제2호 (2012. 7), pp. 3-6

[6] Seongeun Yang,Chang, yeol Choi, Hwangkyu Choi, Design and Implementation of Vehicle Route Tracking System using Hadoop-Based Bigdata Image ProcessingJournal of Digital Contents Society Vol. 14 No. 4 Dec. 2013, pp. 447-454

[7] Young-Gil Kim, Geon-Chul Sin, Su-Kyoung Kim, Collaboration Interface for Hadoop-based Bigdata Processing Platform, 2013 한국정보기술학회 하계학술대회 논문집, pp. 511-516

[8] Wikipedia, online, http://en.wikipedia.org/wiki/Apache_Hadoop

[9] Hadoop, online, http://smallmir.tistory.com/222

[10] Map-reduce, online, http://over153cm.tistory.com/23

[11] 정재화, 시작하세요 하둡 프로그래밍, 위키북스, pp. 13

[12] Hadoop Eco System, online, http://blrunner.com/1

[13] Encored, online, http://www.encoredtech.com/sub/sub1_1.html

[14] 아주경제신문, online, http://www.ajunews.com/view/20140327165054786

[15] 전기신문, online, http://www.electimes.com/home/news/main/viewmain.jsp?news_uid=109643

[16] 오도은, 전력산업에서의 빅데이터 활용 현황 및 전망, Journal of the Electric World, pp. 18-232