공학석사 학위논문

철강산업에서의 프로세스 마이닝 프레임워크의 개발 : U사 MES 중심으로

2015년 2월

부 경 대 학 교 대 학 원

기술경영협동과정

장 종 익

공학석사 학위논문

철강산업에서의 프로세스 마이닝 프레임워크의 개발 : U사 MES 중심으로

지도교수 김 민 수

이 논문을 석사 학위논문으로 제출함.

2015년 2월

부 경 대 학 교 대 학 원

기술경영협동과정

장 종 익

장종익의 공학석사 학위논문을 인준함

2015년 2월 27일

주 심 공학박사 김 영 진 (인)

위 원 공학박사 서 원 철 (인)

위 원 공학박사 김 민 수 (인)

철강산업에서의 프로세스 마이닝 프레임워크의 개발 : U사 MES 중심으로

장 종 익

부경대학교 대학원 기술경영협동과정

요 약

많은 기업들이 경쟁력 확보를 위해 ERP, MES, APS 및 SCM과 같은 다양한 정보시스템에 투자를 해왔으며, 이런 정보시스템들의 활용이 수년에 걸쳐 이루어지면서 기업 내에 축적되고 있는 데이터와 응용 프로그램들에서 생성해 내는 로그의 종류와 양 또한 급격히 증가하고 있다. 특히 로그 데이터의 경우 운영기간에 따라 상당한 규모의 데이터가 시스템 내에 축적되었지만 이를 효과적으로 저장하여 추출 및 분석할 수 있는 도구나 방법론이 미비하여 대부분의 로그가 정기적으로 버려지는 실정이었다. 그러나 최근 프로세스 마이닝 기법을 이용하여 이러한 로그 데이터를 효과적으로 분석하여 잠재적인 비즈니스 가치를 발굴해내는 사례와 연구가 발표되면서 이에 대한 관심이 국내외적으로 높아지고 있는 실정이다. 특히 빅데이터 시대로 접어들면서 대용량 데이터의 저장과 분석을 위한 도구와 방법론이 활발히 연구되면서 이들 분야의 연구 결과를 프로세스 마이닝 분야에 효과적으로 활용할 수 있는 방법이 체계적으로 제시될 필요가 있겠다.

본 논문에서는 로그의 종류를 유형화하고, 빅데이터를 위한 데이터 저장과 분석 도구로써 많이 활용되고 있는 Hadoop과 Hive를 이용하여 프로세스 마이닝 관점에서 이를 저장 및 분석하는 과정을 체계화하고자 한다. 로그의 유형과 분석 상황에 맞게 필요한 정보를 추출하여 이를 Hive상에 저장하고, 분석 주제에 따라 Hive로부터 데이터를 조회 및 정제하여 그 분석결과를 생성해내는 방법을 제안한다. 제안한 방법론을 실제 철강산업의 로그 데이터에 적용하여 DISCO를 통해 분석함으로써, 본 논문의 방법론이 실제 프로세스 마이닝의 분석 방법론으로 활용될 수 있음을 예시하였다.

Keyword: 프로세스 마이닝, 로그 분석, 로그 분석 방법론, 철강산업

<목 차>

제 1 장 서론

제 1 절 연구배경 및 목적

정보기술의 발전과 기업정보 시스템의 다양화로 기업은 점점 더 많은 데이터와 그 데이터를 생성하기 위해 실행된 어플리케이션 로그들이 기하급수적으로 증가하고 있다.

과거 IT 시스템에서 제공하는 정보를 통해 기업이 의사결정을 했었지만 최근 빅데이터에 대한 관심이 고조되면서 하둡, MapReduce, NoSQL 등 분산 처리 기술을 사용한 빅데이터 분석의 사례를 찾아보기 어렵지 않다. 이런 빅데이터 분석은 의사결정을 위해 더 많은 정보 즉, DBMS상에 나타나지 않는 정보를 빅데이터를 사용해 시스템의 미세조정, 의사결정 안내, 기존에 불가능했던 제품 개발 등을 할 수 있는 환경을 기업에 제공한다. 빅데이터는 이런 이점이 있지만 연구에 따르면 조직의 85%가 빅데이터 활용계획이 있으나 17%만이 활용할 수 있는 적절한 역량을 갖추고 있다고 한다.[1] 이런 현실을 반영하듯 대부분의 기업들은 기업에서 데이터 분석과 데이터 생성을 위해 실행되었던 어플리케이션 로그 분석하기는 쉽지 않다.

빅데이터 이슈가 점점 커지고 있는 현시점에서 빅데이터 플랫폼을 활용한 프로세스 마이닝 연구도 활발히 진행되고 있다.

그러나 프로세스 마이닝 분야에서 웹로그 분석이나 보안 로그 분석에 대해서는 많은 연구가 진행되고 있지만 업무 프로그램의 로그 분석분야의 연구는 활발하지 않고 하더라도 주로 단일 형태의 로그가 주를 이룬다.

본 논문에서는 철강산업의 MES에서 발생한 다양한 로그를 유형별로 분류하고 NoSQL의 일종인 Hive에 저장하는 방법을 정리하고 이를 추출하여 프로세스 마이닝 하는 절차를 제안한다.

제 2 절 논문의 구성

본 논문은 다음과 같이 구성된다. 1장은 연구배경 및 목적을 소개한다. 2장은 제안 프레임워크에 필요한 관련 기술을 정리한다. 3장에서는 현재 시스템을 분석하고 제안 프레임워크의 요구사항을 정리 한다. 4장에서는 3장에서 분석된 현재 시스템과 제안 프레임워크의 요구사항을 종합하여 프로세스 마이닝을 위한 프레임워크를 제안한다. 5장에서는 구현된 시스템을 통해 실제 사례를 들어 분석을 수행한다. 마지막 6장에서는 결론과 향후 연구과제에 대해서 제시한다.

제 2 장 관련기술 정의

제 1 절 빅데이터

데이터는 '21세기 원유'라c 불리며, IT·금융·유통 등 다양한 산업 분야의 새로운 패러다임이자 신성장동력으로 부상하고 있고 글로벌 시장 규모는 연평균 27% 성장해 2017년 324억 달러 규모에 이를 것으로 전망하고 있다.[2] 국내 시장의 경우 2015년 기준 2.6억 달러, 20년에는 8.9억 달러로 연평균 24%의 성장을 보일 것으로 한국과학기술정보 연구원은 전망하고 있다.[3]

가트너에서는 빅데이터의 정의는 규모(Volume), 다양성(Variety), 빠른 속도(Velocity)를 특징으로 하는 데이터를 의미하며 지금까지 처리하기 힘든 방대한 데이터를 분석함으로써 의미 있는 정보를 추출하는 분야로 정의 하고 있다.[4]

또 정지선은 빅데이터는 방대한 규모(Volume), 빠른 처리 속도(Velocity), 다양한 형태(Variety), 새로운 가치(Value)의 '4V'로 정의 했다.[5]

이처럼 빅데이터를 보는 관점이나 기관에 따라 달리 정의 되고 있지만 다음과 같은 공통적인 부분을 얘기 하고 있다.

① 지금까지의 데이터에 더해 기존에 분석하기 힘든 데이터 형태

② 기존의 처리 범위를 넘어서는 규모

③ 기존 시스템으로 처리하기 힘든 빠른 데이터 처리 속도

빅데이터의 정의를 간단하게 요약하면 빅데이터란 현재 시스템의 처리 범위를 넘어서는 데이터이며, 기존과 다른 새로운 처리 및 분석 방법이 필요하다는 것이다. [6]

빅데이터의 특성과 효과는 표 1 과 같다.[7]

<표 1> 빅데이터의 특성과 효과

빅데이터특성	효 과
대규모 (Huge Scale)	• 기술 발전으로 데이터 수집, 저장, 처리 능력 향상 • 현실 데이터를 기반으로 한 정교한 패턴 분석 가능 • 데이터가 많을수록 유용한 데이터, 전혀 새로운 패턴의 정보를 찾아낼 수 있는 확률 증가
현실성 (Reality)	• 우리사회 일상에서의 데이터 기록물의 증가 등 현실정보, 실시간 정보의 축적이 급증할 전망 • 개인의 경험, 인식, 선호 등 인지적인 정보 유통 증가
시계열성 (Trends)	• 현시점뿐만 아니라 과거 데이터의 유지로 시계열적인 연속성을 갖는 데이터의 구성 • 과거, 현재, 미래 등 시간 흐름상의 추세분석 가능
결합성 (Combination)	• 의료, 범죄, 환경, 안보 등 타 분야, 이종 데이터의 결합으로 새로운 의미의 정보 발견 • 실제 물리적인 결합 이전에 데이터의 결합을 통한 사전 시뮬레이션, 안전성 검증 분야 발전 가능

김지숙이 언급하고 있는 빅데이터 종류는 다음과 같다.

<표 2> 빅데이터의 종류[6]

정의	설명
정형 (Structured)	고정된 필드에 저장된 데이터 (ex) 관계형 데이터베이스, 스프레드시트
반정형 (Semi-Structured)	고정된 필드에 저장되어 있지는 않지만, 메타데이터나 스키마 등을 포함하는 데이터 (ex) XML이나 HTML 텍스트
비정형 (Unstructured)	고정된 필드에 저장 되어 있지 않은 데이터 (ex) 텍스트 분석이 가능한 텍스트 문서 및 이미지/동영상/음성/데이터

제 2 절 Hadoop

I. Hadoop의 역사

클라우드 컴퓨팅을 위한 대표적인 오픈소스 분산 파일 시스템인 Hadoop의 역사는 1999년도 웹이 폭발적으로 성장하고 있던 시기에 웹페이지로부터 정보를 추출하기 위한 루씬 프로젝트로부터 출발했다. 더그 커팅[1]은 이 기술을 이용해 방대한 양의 텍스트에서 빠르게 정보를 뽑아낼 수 있는 기술을 오픈소스화 했다. 그 후 2002년도 너치 프로젝트[2]를 시작했다. 당시만 하더라도 알타비스타, 라이코스, MS, 구글과 같은 대형 포털을 가진 회사만이 검색엔진 기술을 가지고 있었고, 경제적인 이유로 외부에 공개 하지 않았다. 더그 커팅은 다른 사람들과 공유할 수 있는 검색엔진을 오픈소스로 개발을 위한 아이디어와 2003년 구글 파일시스템(The Google File System)이라는 구글이 발표한 논문에서 많은 아이디어를 얻어 분산파일 시스템인 너치 분산 파일 시스템(Nutch Distributed file System)을 만들기 시작하였다. 구글 파일 시스템과 유사한 구조를 가진 이 파일 시스템은 웹 크롤링과 색인 과정에서 생성되는 굉장히 큰 파일을 생성하기 알맞은 구조를 가지고 있는 파일 시스템이었다. 2004년에 대용량 데이터를 처리하기 위한 프로그램 모델인 맵리듀스(MapReduce)[3]를 포함 시켰다. 2006년에 너치 프로젝트에서 분산 파일 시스템과 맵리듀스를 독립시켜서 새로운 하둡 프로젝트를 생성했다. 이후 2006년도 야후에 합류한 뒤 2008년에 1만개의 하둡 코어를 이용하여 야후 서비스의 색인 제품들이 생성되고 있다고 발표하였고 그 이후 분산처리 환경에서 더 빠르게 처리할 수 있도록 발전하였다.[8]

II. 핵심 Hadoop 컴포넌트

1) Hadoop의 창시자
2) Apache Nutch : 웹 검색엔진
3) Map과Reduce로 된 프로그래밍 모델(http://research.google.com/roundtable/MR.html)

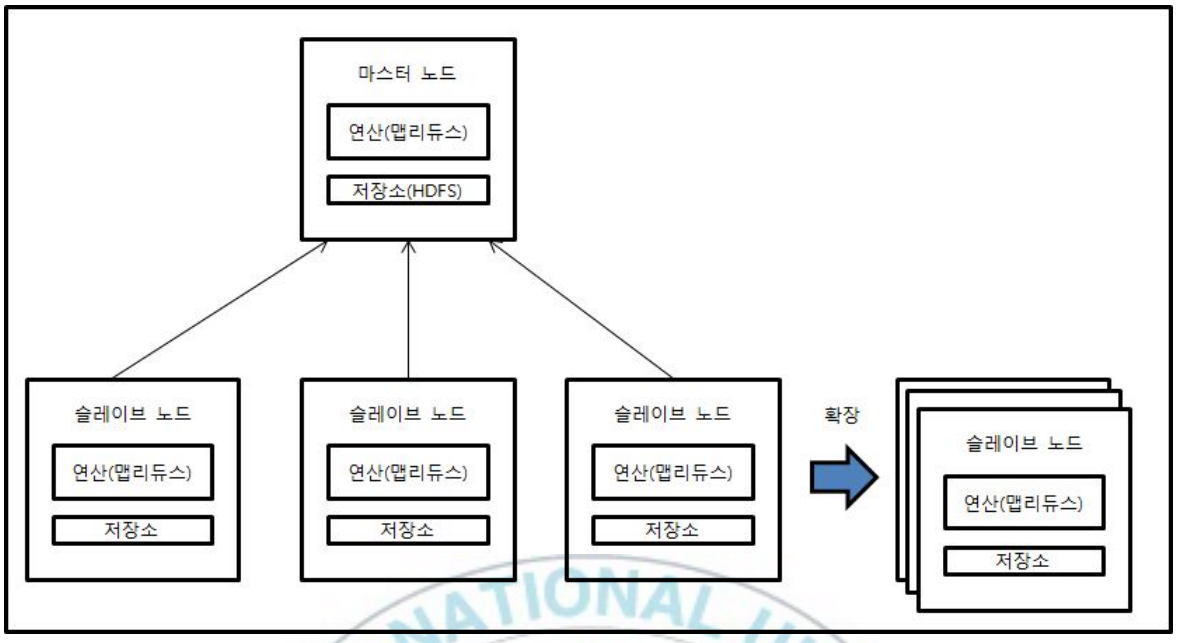
Hadoop은 대용량 데이터를 처리하기 위한 분산 응용 프로그램을 작성하고 실행시키기 위한 오픈 소스 프레임워크이다. Hadoop이 다른 분산 컴퓨팅과 다른 주된 차이점은 다음과 같다.[9]

- 접근(Accessible) : 하둡은 일반 중저가 범용 컴퓨터들로 구성된 큰 규모의 클러스터나 아마존의 EC2[4]와 같은 클라우드 컴퓨팅 서비스에서 실행된다.
- 견고성(Robust) : 하둡은 일반 범용 컴퓨터에서 실행 되도록 의도되었기 때문에, 하드웨어의 빈번한 고장을 가정하고 설계되었다. 대부분 고장에 대해 적절한 조치가 가능하다.
- 확장가능성(Scalable) : 대용량 데이터를 처리하기 위해 노드를 추가함으로써 선형적 확장(스케일 증가가 아닌 스케일 확장)이 가능하다.
- 간단성(Simple) : 효과적인 병렬 코드를 빠르게 작성이 가능하다.

(1) HDFS

HDFS는 구글 파일 시스템(GFS) 논문을 따라 모델링한 분산 파일 시스템이며 대용량 파일 처리에 효과적이다. 이런 쓰루풋(throughput)을 지원하기 위해 큰 블록 크기 및 데이터 로컬리티를 통한 최적화를 활용해 네트워크 입/출력을 줄인다.[10]

4) EC2 : 아마존의 Elastic Compute Cloud(http://aws.amazon.com/ko/ec2)



<그림 1> 고수준 하둡 아키텍처

그림 2는 고수준 하둡 아키텍처를 표현한 그림이다. HDFS마스터는 슬레이브 노드 사이의 저장 공간 파티셔닝과 데이터 저장 위치를 관리하는 책임을 담당하고 맵리듀스 마스터는 슬레이브 노드에서 실행 예약할 연산 작업을 관리하는 책임을 담당한다. [10] 슬레이브 노드는 또한 분산 파일 시스템 환경에서 하나의 파일을 여러 개의 슬레이브 노드에 동일하게 복제하여 만일의 경우 특정 노드의 고장이 났을 때를 대비한다. 그래서 디스크나 서버의 고장 시에도 언제든지 데이터를 사용할 수 있다.[8]

하둡 분산 파일 시스템에서 마스터노드를 네임노드(Name node)라고 하고 슬레이브 노드를 데이터 노드(Data node)라고 한다.

(2) MapReduce

MapReduce는 일반적인 직렬 프로그래밍 기법을 사용하면 며칠 이상 걸릴 작업을 하둡 클러스터의 mapper와 reducer를 이용하여 작성하면 몇 분 만에 작업을 마칠 수 있게 해준다.

맵리듀스 모델은 연산 병렬화, 작업 분산, 비안정적 하드웨어 및 소프트웨어를 다루는 복잡한 요소를 추상화 하여 병철 처리를 단순화하여 분산 시스템의 복잡성에 신경 쓰지 않고도 비즈니스 요구 조건을 처리하는데 집중할 수 있다.[10]

아래 의사 코드는 맵 함수의 논리적인 형태이다.

map(key1,value) -> list<key2,value2>

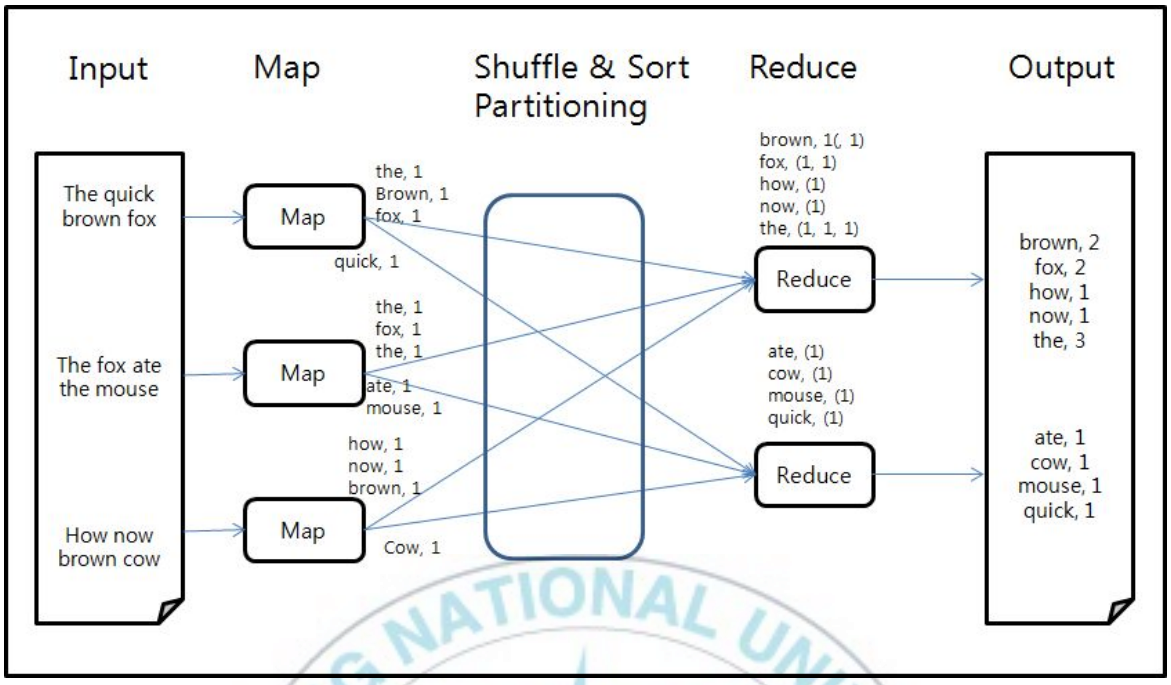
맵 함수는 입력 데이터의 논리적 레코드를 나타내는 키/값 쌍을 입력 값으로 받는다. 파일의 경우 이 값은 한 줄이 될 수도 있고, 입력 소스가 데이터베이스의 테이블일 경우 튜플이 될 수 있다. 맵 함수는 한 개의 입력값 쌍에 대해 0개 이상의 출력 키/값 쌍을 내보낸다. 예를 들어 맵 함수가 필터링 맵 함수이면 특정 조건이 리듀스 함수에서는 충족될 때만 결과를 출력할 수 있다. 또는 한 개의 입력 키/값이 여러 개의 키/값 출력 쌍을 반환하는 역 다중화 작업을 수행할 수 있다.[10][11]

맵 함수에 의해 출력된 리스트는 셔플(Shuffle) 및 정렬 단계를 거친다. 맵 출력/키값 쌍을 수신할 리듀서를 판단하는 파티셔닝 작업과 해당 리듀서에 대해 모든 입력 키가 정렬되게 하는 단계이다.

리듀서의 입력 역시 키/값 쌍이다.

reduce(key2, list<value2>) -> list<value3>

리듀스 함수는 고유 맵 출력키별로 한 번씩 호출된다. 리듀스 출력값은 HDFS 내 파일에 저장하거나, NoSQL 데이터베이스에 행을 삽입/업데이트 하거나, 다른 요구 조건에 따라 임임의 데이터 싱크에 쓸 수 있다.



<그림 2> 단어 빈도수 세기 맵리듀스 예제

아파치 하둡 배포판에 포함되어 있는 단어 빈도수 세기를 예를 들면 입력된 데이터는 3개의 맵 태스크에 의해 키/값 쌍으로 변형된 후 정렬과 파티셔닝 작업을 동시에 수행한 후 셔플과정을 통해 맵 태스크의 결과를 리듀스 태스크의 입력값으로 넘긴다. 리듀스 태스크에서는 입력 값들을 리듀스 하여 최종 결과를 생성한다.[8]

제 3 절 NoSQL, Hive

I. NoSQL

NoSQL은 'Not Only SQL'의 약어로 기존의 SQL만 고집하지 말고, 필요에 따라 SQL에 일부 기능을 포기하더라도 서비스에 특화되어 더욱 적합한 데이터베이스를 선택하여 적용 하자는 것을 기본 아이디어로 삼고 있다. 이에 따라 각각의 NoSQL은 적용하는 서비스에 따라 다른 특성과 장단점을 가진다. 2014년 11월 현재 150개가 있으며 개수는 계속 증가하고 있다.[12]

NoSQL은 Web 2.0 기업이 빅데이터 문제를 해결하기 위해 시작되었다. 기업들이 빅데이터 문제 해결을 위해 처음부터 관계형 데이터베이스를 채택하지 않은 것은 아니다. 관계형 데이터베이스 사용은 매우 큰 트랜잭션 처리에 대한 용량 부담, 매우 큰 데이터 셋에 대한 적은 지연 시간과 그 예측, 신뢰하기 어려운 클러스터 및 노드 환경에서 매우 높은 수준의 고장 감내 및 서비스 운영 능력의 과점에 있어서 문제점을 드러냈다. 높은 성능의 하드웨어를 추가하여 스케일 업 하였지만 곧 한계에 부딪쳤다. 그래서 성능향상을 위한 다양한 기법을 사용하여 극복하려 했지만 한계를 가진 미봉책에 지나지 않았다.

- 관계형 데이터베이스의 스케일 업으로 대응하기 힘든 급격한 데이터의 증가
- 관계형 데이터베이스의 고비용을 감당하기 어려운 업체들의 데이터 관리 정책 전환
- 고장 감내, 재난복구, 지리적 분산에 대한 요구
- 저렴한 클러스터 내 하드웨어 비용에 대한 요구
- 모든 데이터가 트랜잭션이 필요한 것은 아님
- 모든 데이터가 강한 일관성 조건이 필요한 것은 아님

- 서비스에 다른 비정형 데이터에 대한 처리 요구

결국 위의 문제를 해결하기 위해 각 업체들은 자신의 고유한 서비스에 맞는 맞춤형 데이터베이스를 개발함으로써 NoSQL의 시초가 되었다. NoSQL 데이터베이스들은 일반적으로 비기능 요구사항에 해당하는 확장성, 성능, 일관성의 관점으로 다루어지는데 각각의 NoSQL 제품군들의 특성과 내용을 요약하면 다음과 같이 분류할 수 있다.[8]

Key-Value Model

- Key-Value 기반의 단순하고 빠른 Get, Put, Delete 기능을 제공하는 모델로 모든 NoSQL 분류들의 가장 기초적인 모델이다.
- Key-Value 데이터 모델은 매우 단순하면서도 매우 근본적이기 때문에 오히려 매우 강력한 모델이다.
- Key-Value 모델의 가장 큰 단점은 Key 범위 프레세싱이 필요한 경우에 있어 그 적용이 제한된다.

Ordered Key-Value Model

- Key-Value 모델에서 Key 간의 순서에 따른 범위 검색, 순차 읽기 등의 지원으로 단일 데이터의 접근과 함께 특정 범위의 데이터 셋 접근(rang scan)을 지원한다.
- Key-Value, Order Key-Value 모델은 Value에 관한 어떠한 모델링도 제공하고 있지 않기 때문에 프로그램에서 직접 value 처리를 해야 하는 단점이 있다.

BigTable-style Mode

- 행(row)기반의 관계형 데이터베이스와는 다르게 value에 있어 컬럼

(column)을 기본 구성으로 한다.

- BigTable-style 모델은 value 자체를 지속적으로 다차원적인 Map으로 구성한다.(Map의 Map을 지원)

Document Database Model

- BigTable 모델과 비교하여 Value자체를 'Map의 Map' 형태만이 아닌 다양한 스키마를 사용할 수 있고 기본 인덱스를 제공한다.
- JSON, XML 형태로 문서를 구조적으로 저장 가능하다.

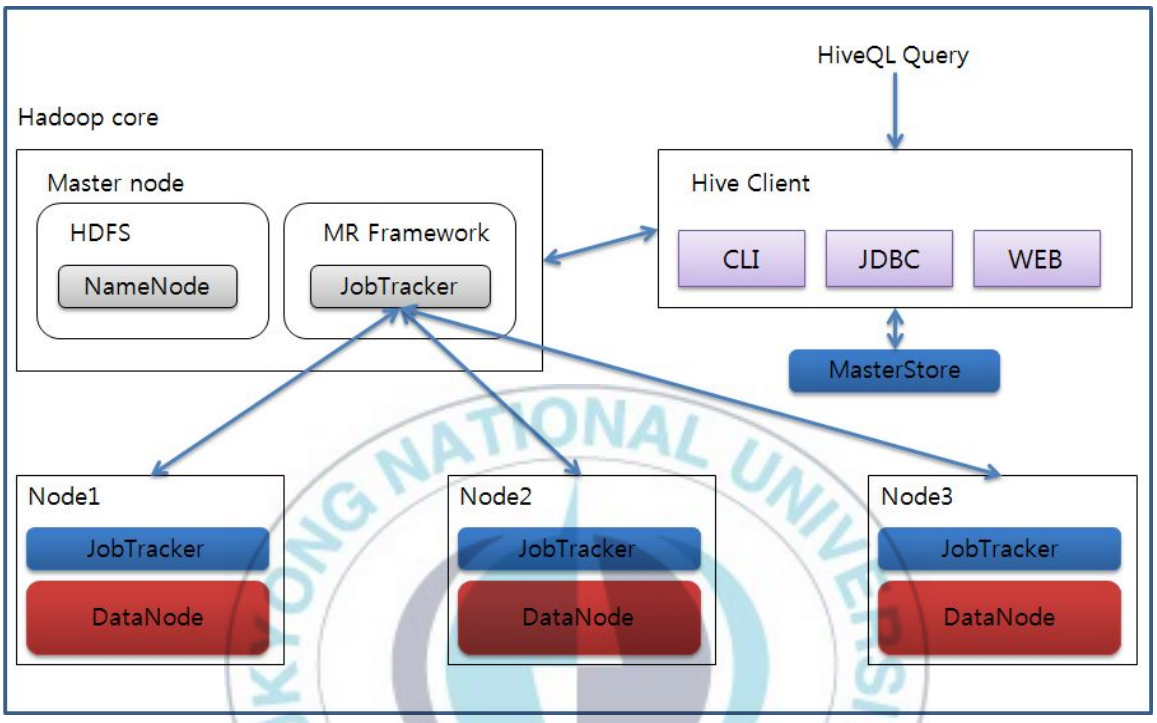
II. Hive

Hive는 NoSQL의 하나로 하둡의 최상위에 있는 데이터웨어하우징 패키지이다. 다수의 사용자 및 대용량 로그 데이터 처리를 위해 페이스북에서 Hive를 만들기 시작했다.[9]

Hive는 기업에서 많이 사용하는 RDBMS와 유사한 환경을 Hadoop상에 구현해 준다. 출현 배경부터 기존 관계형 데이터베이스 인프라를 Hadoop으로 바꾸기 위해 Hive가 출현했다. Hadoop의 MapReduce 프로그래밍 모델이 데이터 처리에 강력하지만 많은 프로그래머나 엔지니어들이 새롭게 배우고 활용하기는 쉽지 않다.

Hive는 MapReduce를 SQL과 유사한 HiveQL이라는 쿼리 언어를 입력 받아 MapReduce를 자체적으로 수행하여 전통적인 DBMS를 사용하던 개발자들이 Hadoop상에서 데이터를 다루기 쉽게 해준다. 그래서 Hive는 전통적인 RDBMS상의 SQL을 사용하여 데이터 처리를 하는 것과 비슷한 HiveQL이라는 쿼리 언어를 제공하여 학습시간이나 학습비용을 줄여 Hadoop으로 접근하기 용이하게 해준다. 현재는 Hive를 이용한 데이터웨어 \하우징 구축 사례가 계속 등장하고

있다.[13][14][15]



<그림 3> Hive 시스템 구성 개요

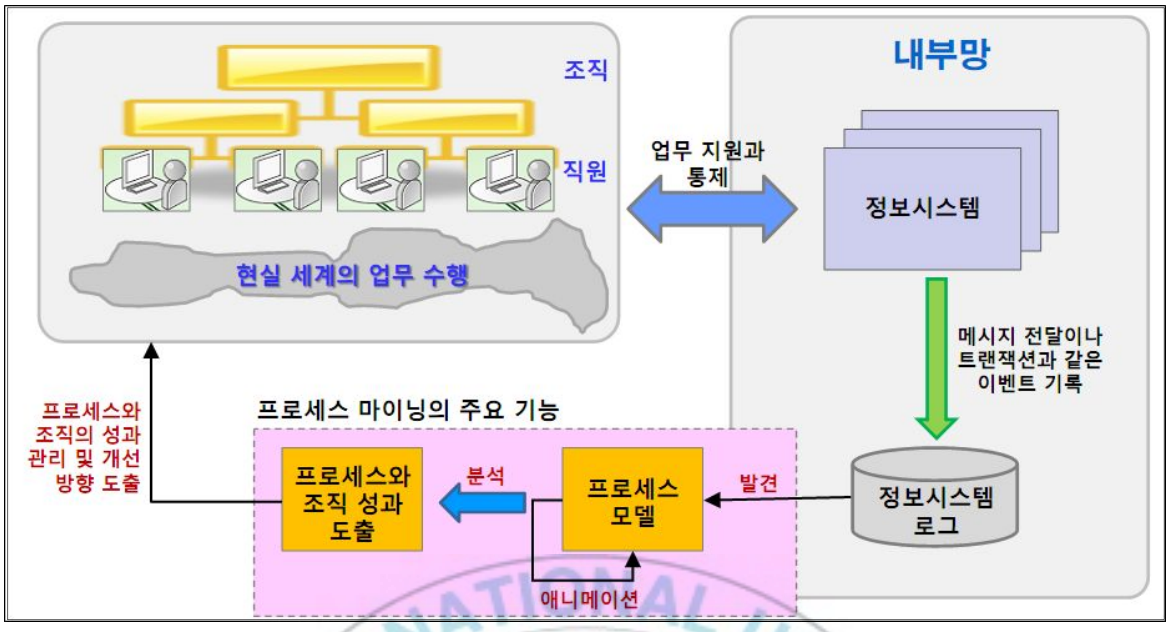
Hive는 HiveQL을 통해 CREATE, DROP, INSERT, SELECT와 같은 명령어를 수행한다. Hive는 구조화된 데이터에 중점을 두어 MapReduce가 갖지 못하는 유용한 특성과 몇몇 최적화 기능을 가졌다. 복잡한 MapReduce 프로그래밍 모델 대신 HiveQL을 사용하여 관계형 데이터베이스를 사용하는 것과 같이 쉽게 데이터 처리를 할 수 있다.

제 4 절 프로세스 마이닝

I. 프로세스 마이닝

프로세스 관리는 기업 경영에 있어 중요한 요소이다. 프로세스 관리를 통해 기업들은 기업 경영을 혁신하거나 효율적으로 개선을 통해 수익을 극대화 하고 있다. 프로세스 관리를 위한 기법들로는 BPR(Business Process Reengineering), 프로세스 또는 조직의 성과 측정을 강조하는 기업 성과 관리(CPM : Corporate Performance Management), 지속적인 프로세스 혁신(CPI : Continuous Process Improvement), 전사적 품질 경영(TQM: Total Quality Management), 그리고 식스 시그마(Six Sigma) 등의 다양한 경영 기법들이 프로세스 성과를 측정하고 개선하는데 활용되고 있다.[22]

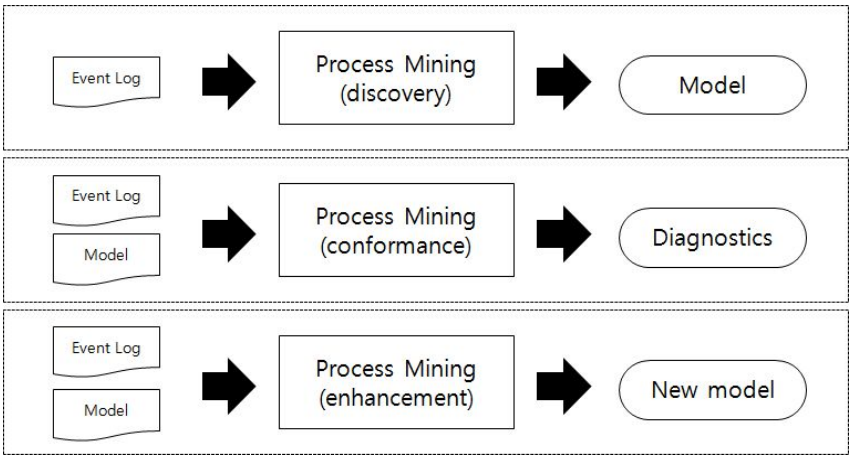
프로세스 분석 기법은 빅데이터 시대를 맞아 새로운 변혁의 시기에 있다. 지속적으로 증가하는 데이터들에 대한 효율적인 분석이 가능하다. 특히 정보시스템의 이벤트 로그 분석은 프로세스에 대한 통찰력을 가져왔다. 기존의 정립되지 않은 프로세스를 이벤트 로그 분석을 통해 시각화하거나 프로세스 관리를 효율적으로 할 수 있는 방법을 제공한다. 그리고 프로세스 전반에 일어나는 병목점이나 문제를 예측가능 하게 되었으며 룰에 근거하여 업무 수행 규정 위반 등의 이상탐지를 통해 대책을 수립할 수 있다.



<그림 4> 프로세스 마이닝을 통한 비즈니스 프로세스 자동 발견/분석[18]

프로세스 마이닝은 MES, ERP, BPM, CRM, SCM 등 다양한 기업의 정보 시스템으로부터 발생하는 빅데이터의 이벤트 로그를 분석하여 '의미 있는 정보를 찾아 내는 것'과 '비즈니스 프로세스를 추출하여 프로세스를 최적화' 하는데 목적이 있다.

프로세스 마이닝 수행은 기법 관점에서 세 가지로 구분이 된다. 첫 번째는 '도출(discovery)'이다. 도출 기법은 이벤트 로그에서 프로세스 모델을 생성하는 기법으로 프로세스 분석에 가장 많이 활용되고 있는 기법이다. 모델이 정립되지 않은 시스템에서 이벤트 로그를 이용하여 빠르게 모델을 생성하여 시각화 할 수 있다. 두 번째는 '적합성 검사(conformance checking)'으로 기존의 프로세스 모델과 생성된 모델을 비교하는 것이다. 적합성 검사는 현실이 모델과 일치하는지 확인하는데 이용될 수 있다. 적합성 검사는 프로세스 모델, 조직 모델, 선언적(declarative) 프로세스 모델, 비즈니스 규칙/정책, 법규 등 다양한 모델에 적용될 수 있다. 세 번째는 '향상(enhancement)'이다. 실제 수행되는 프로세스의 정보를 이용하여 기존의 프로세스 모델을 확장하거나 향상시키는 것이다. [16]



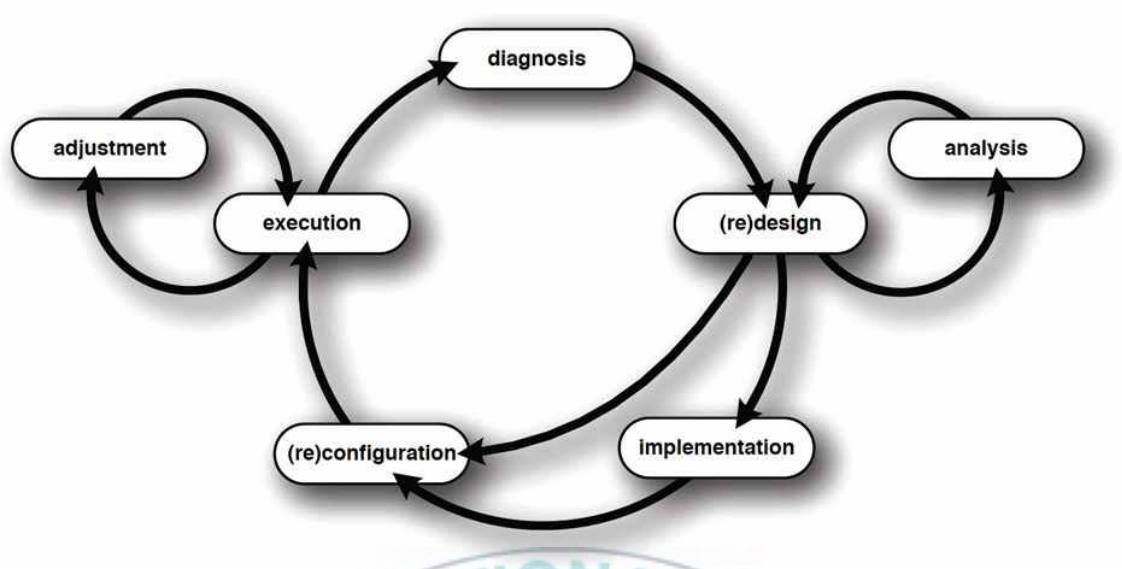
<그림 5> 프로세스 마이닝 분석 타입

프로세스 마이닝을 하기 위한 전제 조건은 수집된 이벤트 로그가 이벤트가 일어난 순서대로 기록되었고 각 Case 별로 저장되어야 한다.[17]

프로세스 마이닝의 효용성은 BPM 라이프 사이클로 설명이 가능하다. BPM 라이프 사이클은 비즈니스 프로세스와 정보시스템이 가지는 라이프 사이클을 7 단계로 나누어서 표현한다.[23]

<표 3> BPM 라이프 사이클 단계별 수행 내용

단계이름	단계별 수행 내용
(재)디자인(re)design	- 새로운 프로세스 모델 생성 - 기존의 프로세스 모델 수정
분석(analysis)	- 후보 모델과 대체 모델들에 대한 분석
구현(implementation)	- "(재)디자인" 단계 이후 - 모델의 구현
(재)구성((re)configuration)	- "(재)디자인" 단계 이후 - 기존 시스템에 대한 (재)구성
수행(execution)	- 디자인된 모델 수행 - 프로세스 모니터링
적용(adjustment)	- 사소한 수정 사항들을 재 디자인 단계를 거치지 않고 수정하며 적용
진단(diagnosis)	- 실행된 프로세스 분석 - 분석 결과를 통해 얻은 정보를 바탕으로 새로운 프로세스 재설계

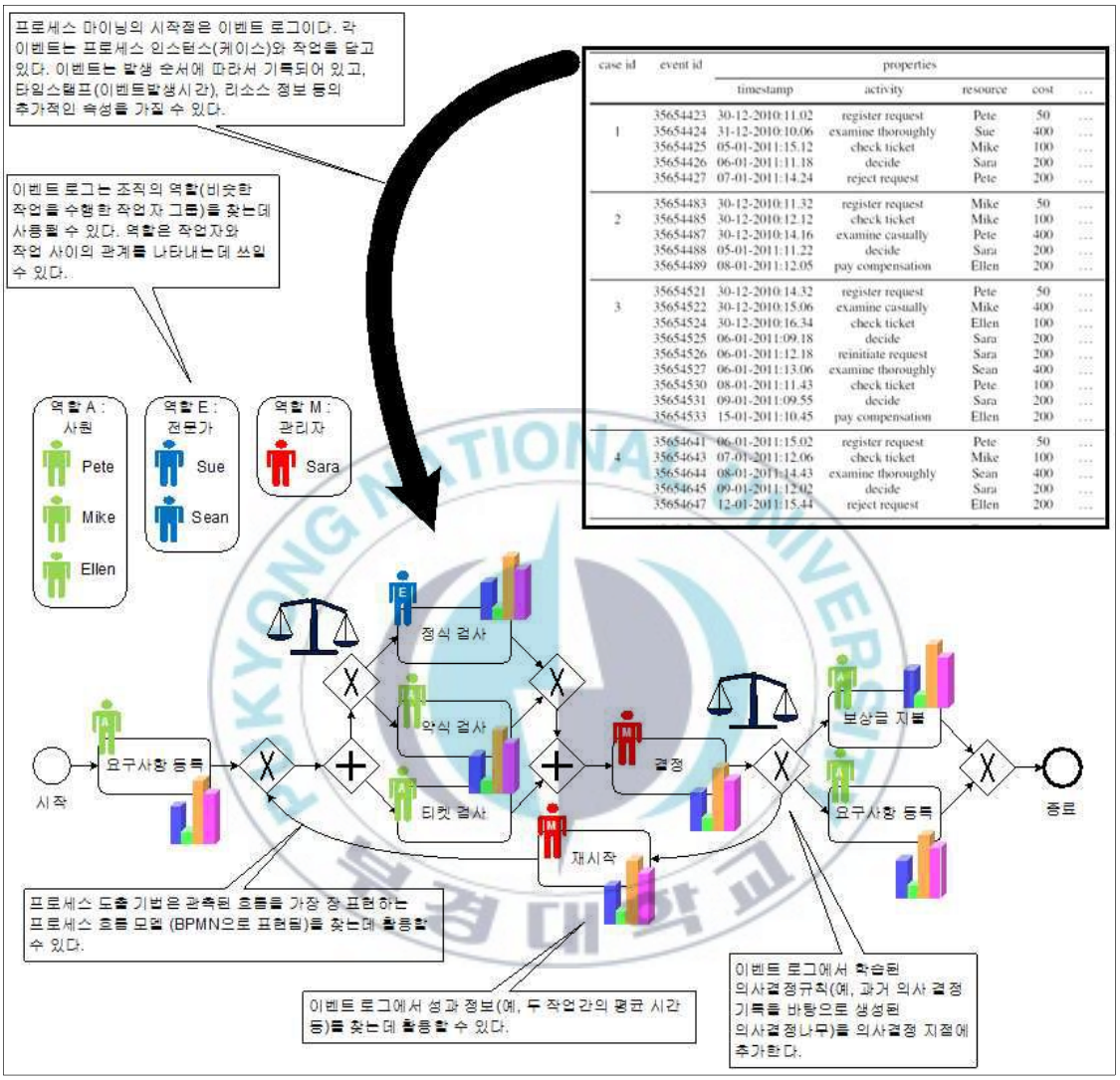


<그림 6> 비즈니스 프로세스와 관련 정보시스템의 BPM 라이프 사이클

그림6은 라이프 사이클로 표현한 도표이다. 프로세스 마이닝은 대부분의 모든 단계에서 유용하게 사용되지만 특히 '진단' 단계에서는 프로세스 마이닝 기법이 매우 유용하다. 적합성 검사 등을 통해 프로세스의 문제점을 찾아내거나 향상을 통해 프로세스 재설계 방향을 제시 할 수 있다. '수행' 단계에서도 과거 데이터 기반으로 학습된 모델을 바탕으로 예측과 추천을 통해 프로세스 운영 지원에 활용될 수 있다.[22]

프로세스 마이닝의 가장 핵심적인 기술은 이벤트 로그에서 프로세스 모델을 도출하는 기술이다. 예를 들어 기업 내에 아직 가시화 되지 않은 프로세스를 찾는데 유용하게 쓰일 수 있다. 기업 프로세스가 명확히 규정되지 않고 정보 시스템에 내재되어 있다면 신규 시스템을 도입하거나 비즈니스 프로세스 개선을 위해 컨설팅 하는 경우에 기존의 업무 시스템을 분석하고 프로세스를 분석하여 AS-IS모델을 만드는데 많은 시간과 컨설턴트와 업무 담당자들의 인력이 필요하다. 만약 정보 시스템을 이용하여 업무를 수행하고 있고 정보 시스템 내에 이벤트 로그가 축적되어 있다면 이 이벤트 로그를 이용하여 프로세스 모델을 생성하면 빠르면 며칠 내로 AS-IS 모델 생성이 가능하다. [22]

병원의 진료 프로세스와 같이 프로세스가 정형화 되어 있지 않은 경우에도 프로세스 마이닝을 통해 프로세스를 가시화 할 수 있다. [23]



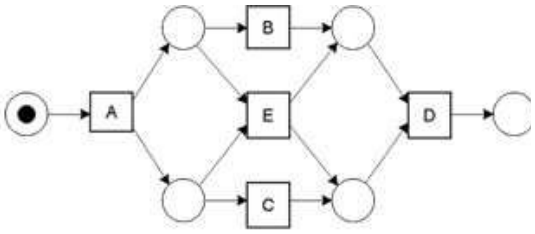
<그림 7> 프로세스 마이닝 예[22]

프로세스 마이닝 알고리즘을 적용하여 누가 어떤 업무를 수행했는지를 자동으로 도출할 수 있다. 이러한 알고리즘에는 가장 처음 개발된 알파 (alpha) 마이닝, 노이즈 처리에 강점을 가지고 있는 휴리스틱 (heuristic) 마이닝, 복잡한 프로세스 모델을 간단하게 볼 수 있는 퍼지 (fuzzy) 마이닝 등이 주로 알려져 있다.[24][25][26]

그림7을 알파 마이닝 기법을 적용하면 Petri-Net으로 표현된다.

Cases	Logs
1	(A,John),(B,Sue),(C,John),(D,Carol)
2	(A,John),(C,Mike),(B,John),(D,Sue)
3	(A,Carol),(E,Mike),(D,Sue)
4	(A,Pete),(C,Carol),(B,Clare) (D,Pete)
5	(A,John),(E,Carol),(D,Clare)

(a) 프로세스 로그



(b) 프로세스 모델

<그림 8> 알파알고리즘을 활용한 모델 도출

또 분석 관점을 달리하여 조직 관점에 대한 분석으로는 이벤트 로그에서 업무 수행자/리소스 사이의 관계를 도출하는 소셜 네트워크 마이닝[27], 업무 관점에서 부서 관계나 팀 관계를 도출하는 조직(organizational)마이닝[28], 업무 수행자/리소스가 어떻게 작업에 할당 되는지에 대한 패턴을 도출하는 스탭 어사인먼트(staff assignment) 마이닝 등이 활용될 수 있다. 또한 업무 성과 관점에서는 프로세스 관점에서 업무의 수행 시간, 병목점 등을 도출하는 페트리넷 기반 퍼포먼스 분석(performance analysis with petri-net), 업무의 수행 패턴을 도출하는 시퀀스 패턴(sequence pattern) 분석, 이벤트 로그의 전반적인 모습 및 패턴 파악에 유용한 도티드(dotted) 차트 분석 등이 사용된다.[22]

II. 프로세스 마이닝 도입 예상 효과

프로세스 마이닝 도입으로 기업이나 조직이 기대할 수 있는 가장 큰 효과는 가장 큰 효과는 업무 수행 및 관리 수준이 향상된다는 점으로, 업무 프로세스의 정형화와 지속적 개선을 통해 기업이나 조직의 역량이 안정화 되고 향상이 되는 것이다.

프로세스 마이닝 도입을 통한 일반적인 효과는 몇 가지로 요약할 수가 있는데 첫째, 프로세스 가시화를 통해 프로세스의 시작부터 종료될 때까지의 모든 프로세스의 연결을 관리 가능하다.

둘째, 도구를 이용한 편리한 모델 생성과 생성된 모델을 시뮬레이션 하여 최적화된 모델 도출을 가능하게 한다.

셋째, 신규인력의 업무 수행 시 교육 및 업무 누수기간 최소화 등을 실현 시키므로 기업의 조직적 역량이 강화 된다.

넷째, 모델을 통한 프로세스의 투명한 관리와 동일한 모델과 도구를 활용한 조직 내.외부적으로 원활한 의사소통이 능력이 향상된다.

다섯째, 자동화된 도구를 사용하여 병목업무의 통계적 파악 및 지속적 개선을 통해 업무 효율성을 높일 수 있을 뿐만 아니라 생산성이 향상 효과를 거둘 수 있다.

본 논문에서 제시한 프레임워크를 사용할 경우 추가적인 효과는 로그 데이터의 자동화된 처리로 빅데이터 수준의 분석 데이터에 대해 빠른 수집 및 처리와 분석 데이터의 효율적인 관리가 가능하다. 또한 여러 분석 관점과 분석방법을 제시하여 프로세스 담당자와 정보 시스템 담당자들 모두를 위해 효율적인 프로세스 개선을 위한 도구를 제공한다.

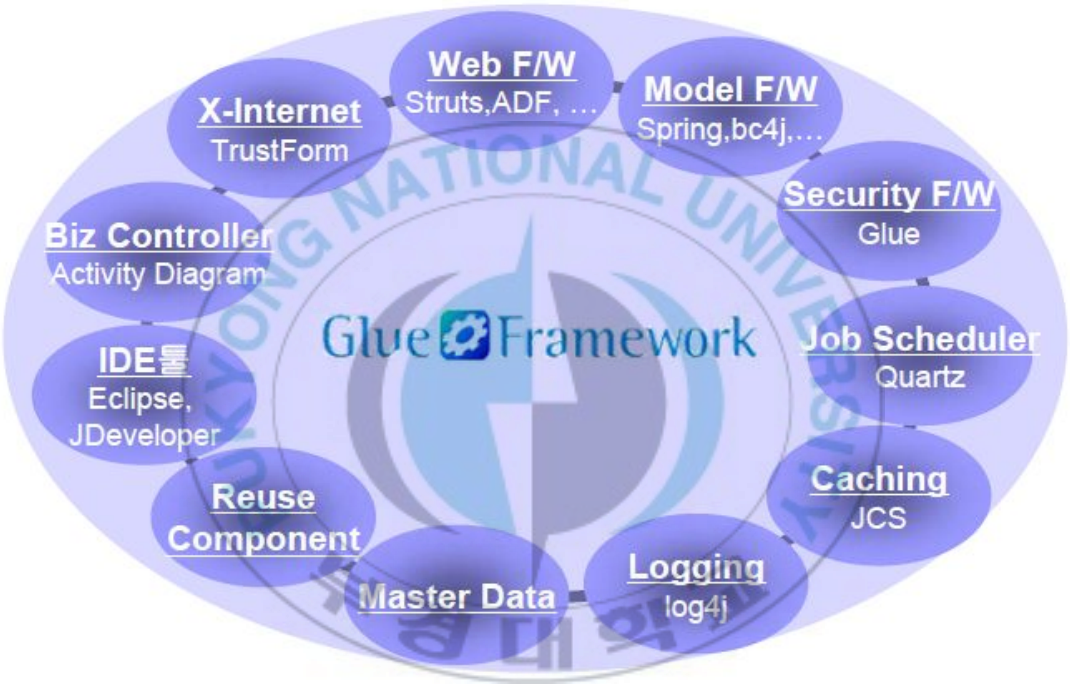
프로세스 마이닝 도입을 통해 예상되는 가장 큰 기대 효과는 급변하는 경영환경에 보다 능동적이고 효과적으로 대응할 수 있는 프로세스의 유연성, 민첩성, 투명성을 확보하는 것이다.

<표 4> 프로세스 마이닝 도입 예상 효과

구분		효과
프로세스 마이닝 효과	프로세스 관점	- 가시화를 통한 체계적인 프로세스 관리
		- 도구를 활용한 모델 생성 자동화
		- 생성 모델을 활용한 최적화 모델 도출
		- 통계적 기법 등의 과학적인 프로세스 분석
		- 지속적인 개선을 통한 업무 효율성 및 생산성 향상
	조직 관점	- 업무 담당자 및 조직의 업무 역량 강화
		- 동일한 시스템 언어 사용으로 의사소통 향상
	시스템 관점	- 어플리케이션 품질 향상
		- 프로세스 그룹화를 통한 프로세스 재사용
		- 자동화 도구로 사용으로 개발 시간, 인력, 자원 최소화로 원가 절감
프레임워크 효과		- 빅데이터 수준의 분석 데이터 관리
		- 분석 데이터의 빠른 수집, 처리
		- 다양한 프로세스 관계자를 위한 분석관점과 방법 제시
		- 범용 도구 제안으로 TCO 감소

제 5 절 Glue Framework

Glue Framework는 POSCO ICT의 표준 J2EE Framework로 플랫폼, 벤더에 독립적이고 UI/Non-UI(Message, Data)처리가 가능한 통합 어플리케이션 Framework이다. 적용분야는 철강, 전자, 호학, 조선 업종에 적용 되었으며 국내에서 대형 철강업체에서 많이 사용되는 Framework 이다.



<그림 9> Glue Framework

주요 기능으로는 비즈니스를 각각의 Activity 단위로 실행 순서와 관계를 Visual하게 설계가 가능하고 SOA를 고려한 POJO기반의 비즈니스 컴포넌트 개발로 Service 단위 재사용이 가능하다. 일반 업무 처리 및 Batch 업무 처리에도 장점이 있다.[19]

제 3 장 제안을 위한 요구사항 및 현재 시스템 분석

프로세스 마이닝은 일회성으로 끝나는 것이 아니라 지속적이고 정기적으로 수행 되는 것이 효과적인 프로세스 관리를 위해 필요 하다. 본 장에서는 지속적이고 정기적으로 프로세스 마이닝을 하기 위한 프레임워크를 제안하고 데이터가 되는 로그를 수집, 정제, 적재, 분석하는 방법을 제안하기에 앞서 프레임워크의 요구사항과 현재 시스템 분석을 한다.

본 논문에서 주안점으로 두고 있는 프로세스 마이닝 프레임워크가 가져야 될 기본 요소는 몇 가지 물음으로 나타낼 수 있다.

1. 누가 무엇을 분석 할 것인가?
2. 로그 데이터 중 어떤 데이터가 분석을 위해 가장 적합한가?
3. 선별된 데이터를 어떻게 획득하는가?
4. 획득된 데이터의 저장 및 관리는 어떻게 하는가?
5. 분석이 필요한 시점에 추출은 어떻게 하는 것이 용이한가?

제 1 절 제안 프레임워크의 요구사항

제안 프레임워크의 특징은 분석 주체별로 분석관점 다양하게 적용해서 분석 대상을 관점에 따라 달리 분석이 가능하다. 분석 관점별로 3가지로 분류하였다.

첫째, 업무 프로세스 관점에서 분석하는 방법으로 기업 시스템에서는 업무 프로세스가 수행되면서 발생하는 직간접적인 데이터가 DBMS에 저장이 된다. 이 로그는 현업담당자가 필요할 때 수시로 분석하는 일상적인 데이터 범주이다. 이 데이터는 주로 업무 진행 또는 기업 활동에 필요한 레포트를 생성하기 위해 사용되는 데이터이다. 이런 실적 데이터도 조금만 가공하면 프로세스 마이닝에 좋은 데이터로 사용될 수 있다. 업무 프로세스 분석에 사용되는 데이터는 생산실적 데이터를 이용한 공정 라우팅 분석과 물류 흐름 데이터를 이용한 야드 치장 이력, 동간 이송 이력, 동내 이적 이력 분석이 있다.

<그림 10> 제안 프레임워크 특징

생산실적 데이터와 물류흐름 데이터를 통합하여 전체적인 공장 내 생산/물류 흐름을 한눈에 파악이 가능하다. 논문 5장에 생산실적 데이터와 물류 데이터를 통합하여 분석한 사례를 수록하였다.

둘째, 서비스 단위 분석은 비즈니스 프로세스가 정보 시스템에 적용되어 정보 시스템 상에 실행 결과로 나온 데이터나 서버에 텍스트 형태로 저장된 어플리케이션 로그를 분석한다. 서비스 단위 분석은 정보 시스템 관점으로 접근하기 때문에 통신 전문 로그 또는 어플리케이션 실행 로그를 대상으로 수집한 데이터에 대해서 분석을 수행 하고 분석 주체는 시스템 분석가와 전산 운영팀에서 수행한다. 서비스 단위 분석의 주목적은 정보 시스템 상의 프로세스 복구, 프로세스간의 적합도 분석, 프로세스 병목점 분석 등으로 정보 시스템의 프로세스 개선을 위해 사용된다.

셋째, 액티비티 단위 분석은 서비스를 이루는 각 액티비티 항목에 대해 분석하는 것이다. 정보 시스템의 서비스를 이루는 가장 기본단위의 분석으로 프로그램 오류 추적, 액티비티 수행 속도를 분석하여 튜닝대상 발굴, 프로그램 설계문서 복구에 사용된다. 액티비티 분석은 액티비티가 많아서 전체로 분석하기는 힘

들고 몇몇 서비스를 선정하여 분석하는 것이 바람직하다.

업무 프로세스, 서비스, 액티비티의 관계는 표 1에 예를 들었다. 제품 생산이라고 하는 하나의 업무 프로세스는 내부적으로 재료 보급, 재료 투입, 검사실적 등의 서비스로 이루어져 있고 각 서비스는 다시 여러 개의 액티비티로 이루어져 있다.

<표 5> 업무프로세스, 서비스, 액티비티 구분의 예

업무프로세스	서비스	액티비티
1. 제품생산	1. 재료 보급	1. 재료 위치검색 2. 재료 보급위치 설정
	2. 재료 투입	1. 재료 투입위치 설정 2. 재료 투입시각 설정 3. 재료 투입실적 등록.
	3. 검사 실적	1. 생산시각 설정 2. 작업일자, 작업조 설정 3. 새로운 코일ID 발번 4. 코일공통 등록 5. 제품실적 등록 6. 부적합실적 등록 7. 타 시스템 검사실적 전송 8. 태그 발행 9. 재료진행 실적 생성 10. 생산스케줄 조정
	4. 작업 실적	1. 작업실적 항목계산 2. 작업실적 등록

제 2 절 요구 기능

I. 대용량 처리 및 확장 가능성

비정형이고 대용량인 로그의 특성을 고려하여 선형적 확장이 가능하고 분산 처리가 가능한 형태로 구현 되어야 한다. 오픈소스 프레임워크인 Hadoop과 기존 업무에서 SQL언어를 사용하는 프로그래머들이 쉽게 적응 가능한 쿼리 언어 형태의 Hive를 사용하여 대용량 처리 및 적은 교육비용으로도 구축 및 활용 가능해야 한다.

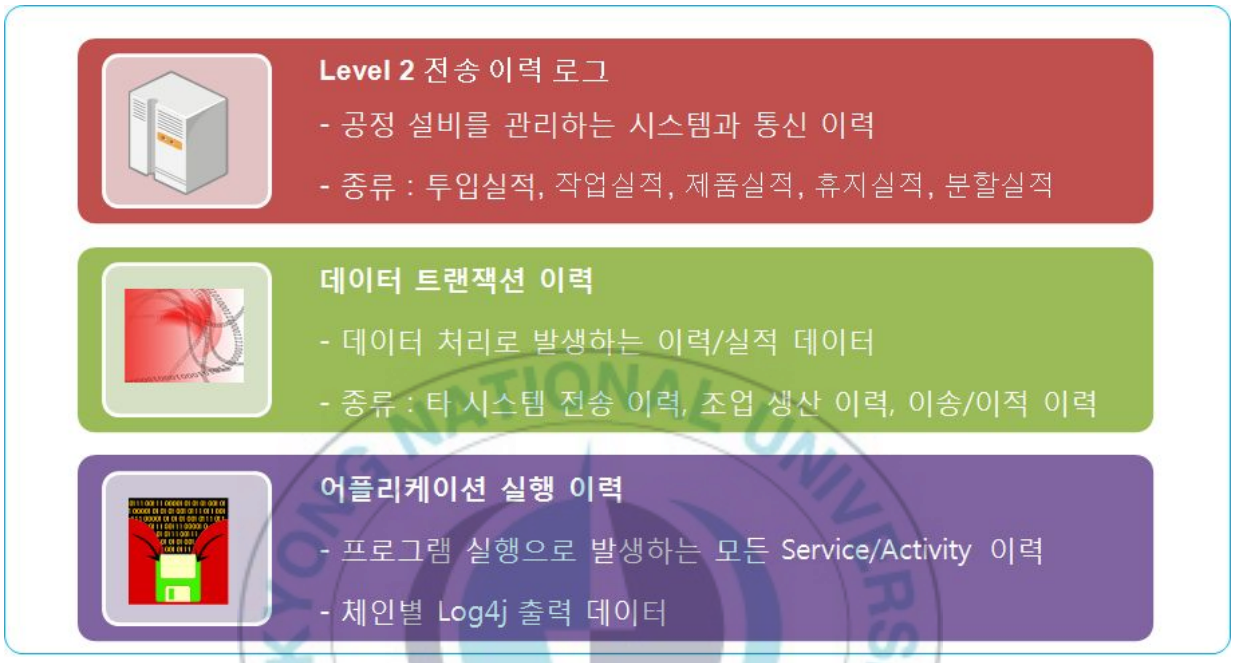
II. 다양한 로그 처리

제안 프레임워크는 특정 포맷의 로그가 아닌 통신전문, 실제 업무 프로세스의 결과로 얻어진 생산이력, 이송/이적 이력을 포함한 DBMS의 프로세스 이력, 정보 시스템이 수행되면서 텍스트 형태의 이력 등 다양한 로그를 추출하고 분석할 수 있어야 한다. 특히 정보 시스템 수행 이력은 처리과정에서 구조적인 로그를 생성하고 업무 프로세스 추적에 대한 지원도 가능해야 한다.

III. 빠른 데이터 추출

업무의 특성상 다양한 제품군, 제품형태에 대해 필요에 따라 빠른 추출을 위해 분석 데이터를 선택하여 추출이 가능해야 하고 저장공간의 효율성과 저장주기에 대한 효율성도 같이 고려하여야 한다.

제 3 절 제안 프레임워크의 로그 분석 대상



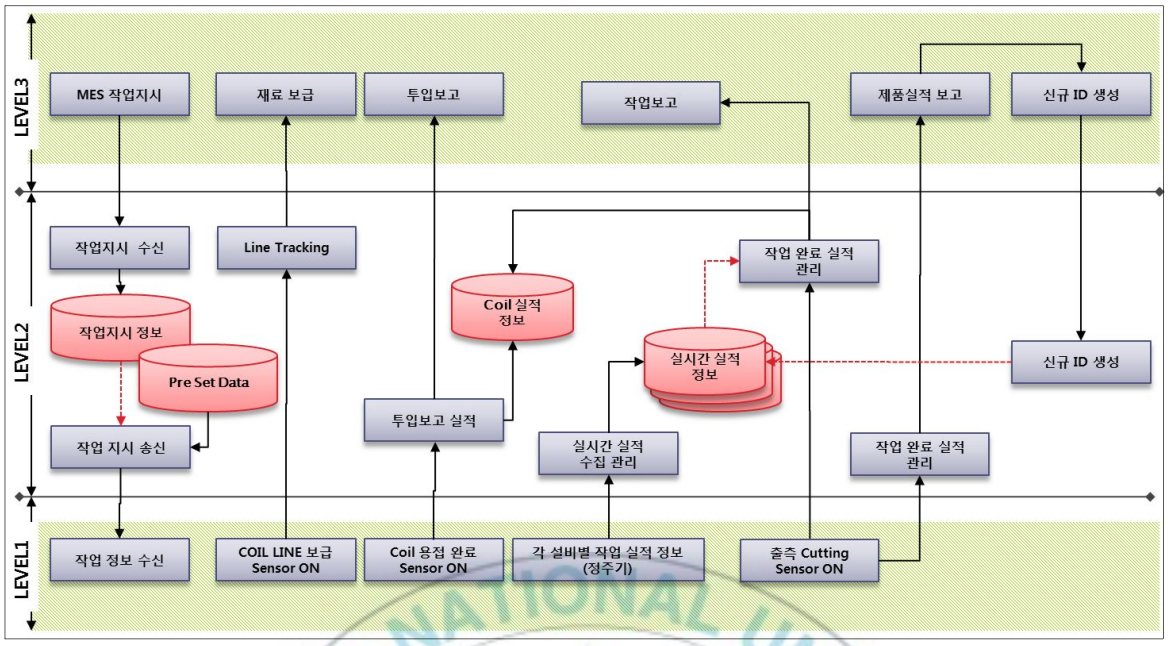
<그림 11> 로그 분석 대상

그림 11은 U사가 MES정보 시스템에서 사용 중인 Glue Framework 내의 업무에 관련된 로그를 정리한 것이다.

제안 프레임워크에서 사용될 로그를 발생위치나 분석목적에 의해 분류하면 크게 3가지로 부류할 수 있다.

I. Level 2 통신 전문 로그

Level 2 전송 이력 로그는 공정관리 시스템인 Level 와 MES간의 소켓 또는 EAI를 통해 통신한 이력으로 형태는 구분자 없는 텍스트 형태로 되어 있다. Level 2 시스템은 MES로부터 작업지시를 받아 관리하고 PLC나 Level 1 시스템으로부터 받은 시그널을 조합하여 MES에서 처리할 실적 이벤트를 생성하여 전송한다.



<그림 12> Level 간 통신 전문(예시)

그림 12는 U사의 칼라코팅공정 중 하나의 공정 프로세스를 표현한 것이다. 공정마다 처리방식과 설비 및 LEVEL 2 제조사에 따라 차이는 존재하겠지만 제품을 생산하기 위한 일반적인 절차는 다음과 같다.

① LEVEL3에서 작업지시를 받아서 각 장비의 설정 값인 프리셋(Preset) 데이터와 함께 LEVEL 1에 작업지시를 내린다.

② 재료가 라인에 보급이 되면 LEVEL2에서는 LEVEL1으로부터 보급신호를 받아서 Tracking 데이터를 관리하고 정주기로 LEVEL3에 보내주는 Line Tracking 전문을 통해 MES에서는 재료 보급 서비스를 수행한다. 재료보급이 되면 공정에 재료를 투입할 준비가 끝난다.

③ 코일을 투입하기 위해서는 POR(Pay-Off Reel)에 코일을 장착하고 공정에 먼저 투입된 코일과 용접을 한다. 용접점이 센서를 통과하면 LEVEL1에서 용접완료 이벤트를 LEVEL2로 전송하고 LEVEL2에서는 이를 받아 LEVEL2처리 후 MES로 투입 전문을 전송한다. MES에서는 투입전문 수신 후 MES 투입 서비스를 실행한다.

④ 작업 중인 재료코일의 작업환경(온도, TENSION 등)의 센서 값을 LEVEL1을 통해 취합하여 LEVEL2로 정주기(보통 10초 내외)로 전송한다. LEVEL2에서는 이 정보를 취합하여 출측 TR(Tension Reel)에 감기고 있는 제품 코일이 분할되는 시점에 수집된 작업 실적을 평균처리 투입된 코일 단위로 MES로 전송하고 생산된 코일 단위로 제품실적을 전송한다.

⑤ 제품실적을 수신 받은 MES는 제품실적 서비스를 실행 후 출측에 새로 생긴 코일에 대해 신규 제품 코일ID를 생성하여 LEVEL2에 송신한다. LEVEL2는 신규 제품 코일ID를 실적정보에 저장한다.

이 과정에서 LEVEL2와 MES간 통신 전문이 MES 전문 송수신 테이블에 저장이 된다.

통신전문은 MES 전문 송수신 테이블의 MSG 칼럼에 저장이 된다. 통신전문은 보통 전문별로 포맷이 다르게 구현 되어 있다. 30개가 넘는 공정 중에서 A6 공정을 중심으로 LEVEL 2전문의 행태를 수록했다.

B47RA6022000LA62000M4720141023080216 000211 A6200200111201410230802 16 QA02284 VAK831Y 22340018851116500372201410230801 40
B47RA6032000LA62000M4720141020044119 000405 A6200300305201410200441 19 QA03243 V8EA601 V8EA6019 0735102239W09903A609J 12120058
B47RA6042000LA62000M4720141023081526 000304 A6200400204201410230815 26 QA02285 VAK831X VAK831XA VAK831XA 37A14GA4ZH15AL4130A00380111700682022280049981665300508005080000120141023061033201410230815260012400000
B47RA6052000LA62000M4720141024191943 000868 A6200500768201410241919 43 QB01724 VAL8304 38A14FAXZTA5JUT031A00452092100042001372000010041801511201410241847292014102419194200032001100003000030000300041600708008320000000050000000118400000000000000000570004000136001200002900120000290000000000000000000000000000040000600001000030000300004000 01- 0006000900002900010000900002900043000490006000064006020061300807008250000000000000000000000000000000000000000000000000000000000000000000000000000000000000151000670 0000000680000000067000000000000000000210000500023900060002660008000266000600000000000000000000000000000000000000000000000000000000000000000000000000000000000013000000 0000 0000 0000 0000 0000
B47RA6062000LA62000M4720141022062837 000194 A6200600094201410220628 37 QA03342 V9H920LA V9H920LA 0030800308
B47RA6072000LA62000M4720141027002121 000489 A6200700389201410270021 21 02014102623050420141026231739000122014102623173 9F1002 VAKA62U V9QA61W 000LAMINA PRESS/R 오염 因 L-STOP
B47RA6082000LA62000M4720141020080805 000551 A6200800451201410200808 05 VAB7701 V9RA80P VAB7704 VAB7704 DUMMY06662C DUMMY06662C 0002500025000250002500001

<그림 13> Level 2 전문(예시)

<표 6> 프로세스 마이닝 항목과 전문 매칭

구분	전문칼럼	시작위치	글자 수
Case ID	코일 ID	전문마다 위치 다름	7
Activity	전문 ID	1	8
Timestamp	전문전송시각	23	14
Sender	송신시스템	13	3
Receiver	수신시스템	20	3

표 6은 프로세스 마이닝 항목과 전문을 서로 관계를 지어 표시한 표이다. Case ID가 되는 Coil ID를 제외한 나머지 항목은 전문의 헤더부분에 위치하고 있어서 전문에 상관없이 동일한 위치에 정보들이 있다. 하지만 Coil ID는 전문 내용부분에 위치하고 있어 각 전문마다 위치가 다르다. 그러므로 Coil ID는 전문마다 위치를 지정해야 한다.

로그 수집대상은 코일ID 유무와 프로세스의 중요도에 따라 A6공정에서는 8개의 전문만 대상으로 했다. 다른 공정들도 마찬가지로 같은 방식으로 결정 했다.

<표 7> A6 공정의 전문

전문ID	전문명	로그 수집	코일ID위치	송신	수신
B47RA600	작업지시 요구 수신	O	150	LA6	M47
B47RA602	투입 실적 수신	O	148	LA6	M47
B47RA603	입측 분할 수신	O	148	LA6	M47
B47RA604	제품 실적 수신	O	184	LA6	M47
B47RA605	작업 실적 수신	O	148	LA6	M47
B47RA606	출측 중량 계근 실적 수신	O	184	LA6	M47
B47RA607	휴지실적 수신			LA6	M47
B47RA608	Tracking 수신			LA6	M47
B47RA609	조교대정보 수신			LA6	M47
B47RA613	L3CodeRequest 수신			LA6	M47
B47RA621	입측 REJECT 수신	O	148	LA6	M47
B47SA601	작업지시 송신	O	152	M47	LA6
B47SA602	복수오더지시 송신			M47	LA6
B47SA603	New COIL ID 송신			M47	LA6
B47SA606	L2검사실적 송신			M47	LA6
B47SA607	L3작업실적 송신			M47	LA6
B47SA608	L3Code정보 송신			M47	LA6

II. 데이터 트랜잭션 이력

정보 시스템에서 서비스가 수행된 결과가 DBMS 내에 각 테이블에 저장이 된다. 프로세스 마이닝을 위한 실적은 투입실적, 검사실적, 이송/이적 실적이 있다.

투입실적은 공정에서 생산을 시작하기 위해 이전 코일과 현재 코일의 용접이 끝난 후 생산이 시작되면 투입실적을 한다. 투입실적은 공정에서 코일이 생산이

시작되었다는 것을 의미한다. 프로세스 마이닝을 위해 중요한 정보는 코일ID, 투입시각 이다.

공정에서 처리가 된 코일은 검사실적을 하게 되는데 이때 새로운 코일ID를 부여 받는다. 하나의 투입된 코일에 대해 공정 출측에서 여러 개의 생산코일이 나올 수 있다. 이렇게 나온 코일들은 모코일과 다른 새로운 코일ID를 부여 받는다. 그러므로 투입실적의 코일ID와 생산된 코일ID는 부모 자식의 관계를 가지는 형태이다. U사는 하나의 제품을 생산하기 위해 많게는 10개 이상의 공정을 거치는데 공정에서 처리 될 때 마다 새로운 ID를 부여 받으므로 최초 원자재가 투입된 이후 중간 제품과 최종 제품의 코일ID는 몇 십 개가 존재 할 수 있다. 그러므로 로그 수집과정에서는 각 실적에 해당하는 코일ID로 저장하고 추출 시에 코일 이력을 통해 관계를 맺어 추출해야 한다. 검사실적에서 중요한 정보는 투입실적과 마찬가지로 코일ID, 검사시각 이다.

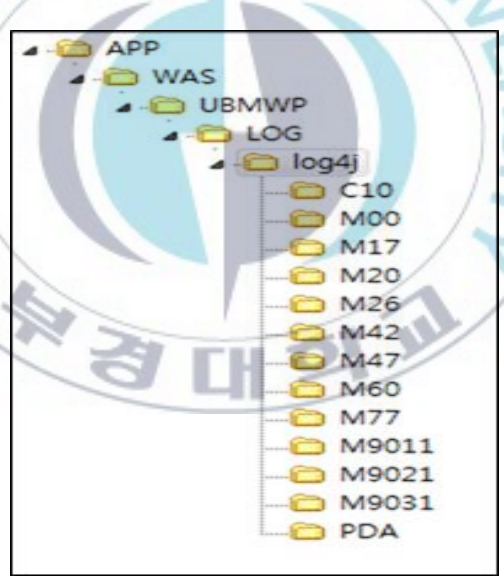
U사는 다양한 냉연, 도금, 도장의 철강 제품을 생산하기 위해 공정 라우팅도 복잡하게 되어 있는데 이런 이유로 인해 공장 내 물류가 복잡하게 처리된다. 생산을 위한 동간 이송, 창고 관리를 위한 이적 등의 다양한 작업이 빈번하게 이루어지기 때문에 물류 흐름 분석 또한 원가절감을 위해 중요한 요소가 된다. 관리되는 치장위치는 동, 열, 행, 단의 4단계 주소 구조를 가진다. 수집 데이터는 공장 내 물류비용, 재고비용의 대부분을 차지하는 동간 이송, 이적을 대상으로 한다. 동간 이적, 이송은 DBMS에 저장된 이벤트로 보고 이송, 이적, 생산 행위가 일어나는 사이의 시간은 재고시간이 된다.

코일 이력은 Hive저장소에 적재될 데이터는 아니지만 Level 2 전문 로그와 데이터 트랜잭션 이력 로그를 추출하기 위해 사용된다. 제안 프레임워크는 로그 이벤트를 분석자가 필요시에 다양한 조건으로 추출하는 하는 것을 목표로 하고 있다. 모든 조건을 Hive에 적재해서 Hive상에서 입력된 조건대로 분석하면 좋겠지만 다양한 조건을 처리하기 힘들고 제한된 저장공간을 효율적으로 사용하기 힘들다. 본 프레임워크에서 제안하는 방법은 Hive 저장 공간에는 로그들의 이벤트만 저장하고 분석자가 분석을 요구하는 시점에 MES에 있는 정보와 코일 이력으로 필터를 만드는 방법을 사용한다.

III. 어플리케이션 실행 로그

UI(User Interface) 프로그램 또는 전문처리나 Batch Job을 처리하기 위한 NUI(Non UI) 프로그램에서 실행한 업무 프로세스는 WAS를 통해 서비스가 실행된다. 어플리케이션 실행 로그는 서비스를 구성하고 있는 Java Class로 작성된 액티비티들의 실행 이력들이 Log4j를 통해 파일로 출력된다.

Glue Framework는 어플리케이션 실행 로그를 출력하기 위해 Log4j로 서비스의 시작과 끝, 액티비티의 시작과 끝, SQL과 파라미터, 그 외 클래스에서 필요에 따라 출력하는 정보를 텍스트 형태로 로그를 출력한다. 물론 DB 테이블에 로그를 저장 하는 방법도 지원하지만 용량문제나 관리의 어려움으로 인해 잘 사용되지는 않고 파일 출력 방법을 주로 사용 한다. U사도 로그를 파일로 출력하는 방법을 사용한다.



<그림 14> 로그 파일 위치 구조

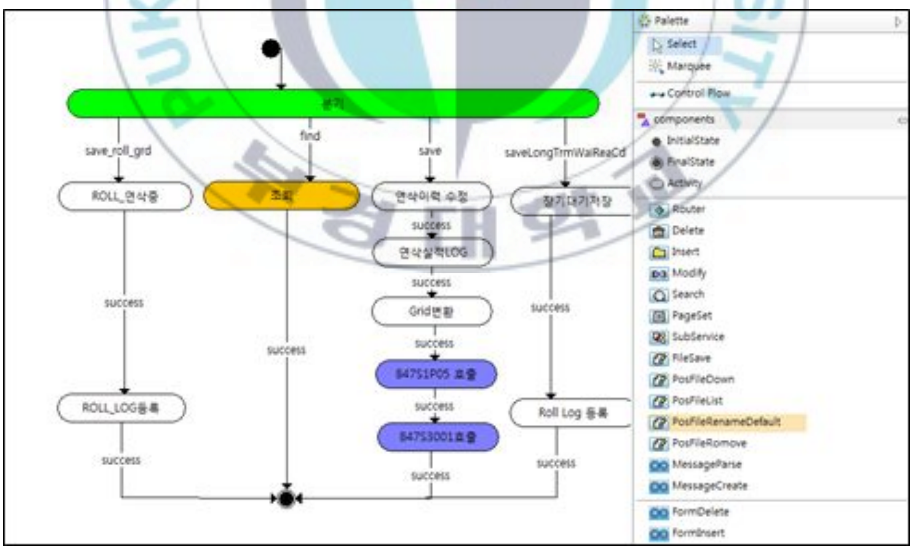
로그 파일의 위치는 업무ID별로 디렉토리에 저장되며 분단위로 저장되게 되어 있다.

```
<appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
        <param name="encoding" value="UTF-8"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern"
                 value = "%d %-5p - [%t]%C||%m%n"/>
        </layout>
</appender>
```

<그림 15> log4j.xml 일부

log4j.xml 의 일부분이다. 여기서는 ConsoleAppender라는 클래스를 통해 로그가 파일로 출력이 된다.

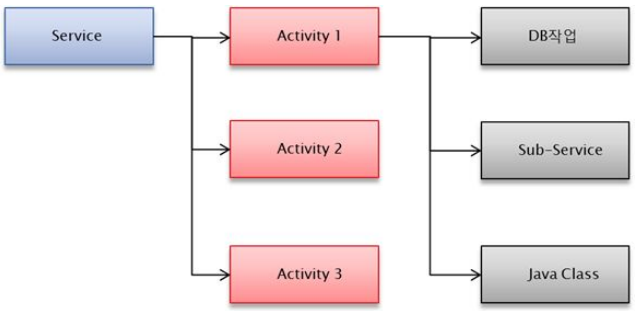
GLUE는 스프링 프레임워크 기반으로 DB연동, 트랜잭션 처리를 GLUE 프레임워크에서 확장하였다. 그림 16은 GLUE 프레임워크의 Activity Diagram의 예이다. Activity Diagram이 하나의 서비스가 되고 도형들은 서비스를 이루는 각각의 Activity 들이다. 연결되는 선은 흐름의 제어를 나타낸다. GLUE는 이 Activity Diagram으로 서비스를 생성하고 생성된 서비스는 WAS를 통해 실행된다.



<그림 16> Glue Framework Active Diagram

GLUE는 Activity Diagram에서 정의한 Activity 실행 순서대로 수행이 된다. WAS에서 해당 서비스를 호출하면 최초 시작점부터 연결되어 있는 Activity를

찾아 조건에 맞게 수행이 되는데 서비스와 액티비티의 시작 시점과 종료시점에 로그를 저장 하도록 되어 있다.



<그림 17> Glue Framework의 로그 구조

그림 17는 Activity Diagram을 개념적으로 표현한 그림이다. 서비스 하부에 Activity가 올 수 있으며 Activity는 보통 Java Class와 맵핑이 된다. Java Class가 DB관련 작업을 하는 클래스이면 DB작업내용을 가지며 Activity가 다른 서비스를 다시 호출하는 구조이면 Sub-Service로 맵핑이 된다. Java Class에서 출력되는 모든 메시지도 Log4j를 통해 로그 파일에 출력이 된다. 요약을 하자면 서비스 아래 일반 Java Class, DB작업 Java Class, Sub-Service 호출 Java Class가 위치하는 구조이다. 그 외에도 오류가 발생했을 때 덤프출력하거나, 보안을 위해 유저 접근 제어, Audit 항목 등이 부가적으로 기록이 된다.

Audit 항목에는 사번 정보가 들어간다. 서비스가 호출이 되면 서비스 시작 정보를 출력하고 서비스 내에 액티비티들을 수행한다. 서비스 시작 로그에서 필요한 추출 정보는 기록일시, WAS실행 쓰레드번호, 서비스ID의 정보이며 기록일시는 서비스 시작시간으로 시간설정을 하고 WAS실행 쓰레드 번호를 이용하여 시간 순으로 뒤섞인 로그를 서비스별로 정렬한다. 서비스ID는 어플리케이션 실행 로그에서 가장 중요한 서비스를 구분하는 구분자로 사용 된다.

액티비티에서 추출할 정보는 서비스와 마찬가지로 기록일시, WAS실행 쓰레드 번호, 서비스ID와 추가적으로 액티비티의 ID를 추출한다. 액티비티는 DB작업, Java Class, Sub-Service등으로 분기하여 수행된다.

DB작업은 반드시 SQL Query가 따라오는데 '['로 시작하여 ']'를 만날 때까지

여러 줄에 걸쳐 출력된다. SQL 쿼리는 첫 번째 줄을 제외하고 두 번째 줄부터는 연속된 쿼리 내용만 출력이 되므로 WAS 실행 쓰레드 번호와 서비스로 직렬화가 되지 않으면 쿼리를 추출하기 힘들다.

Java Class를 쓰는 이유는 대부분 계산을 위해 쓰는 경우가 많으나 특별하게 Java Class 내에서 코딩으로 SQL Query 수행이나 Sub-Service를 수행하는 경우도 있다. 이 경우에는 Glue Activity Diagram에는 나타나지 않지만 로그 처리를 통해 프로세스 마이닝 작업을 통하면 쉽게 찾을 수 있다.

Sub-Service로 분기가 되면 Sub-Service가 또 하나의 서비스가 되고 Sub-Service내의 모든 액티비티들의 작업이 끝나면 호출한 액티비티로 다시 돌아오게 된다. 액티비티 작업이 끝나면 '액티비티 끝' 로그를 출력한다. 다시 제어는 서비스로 넘어가고 서비스는 다음 액티비티가 있는지 판단 후 다음 액티비티가 있으면 액티비티를 모두 수행할 때까지 액티비티를 수행한다. 모든 액티비티 수행이 끝나면 '서비스 끝' 로그를 출력하고 서비스는 종료 된다.

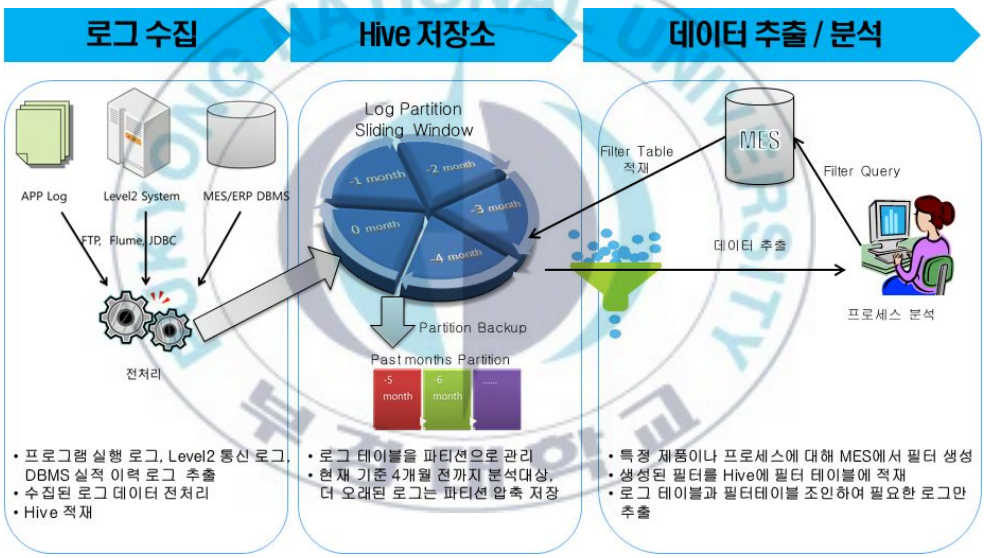
<표 8> 로그 항목의 의미

종류	로그 예제	설명
Audit	9월 22 07:24:00 오후 DEBUG [[ACTIVE] ExecuteThread: '6' for queue: 'weblogic.kernel.Default (self-tuning)'] - PosAuditAttributes.a(246) | Default Audit info : [ObjectType=A][ObjectId=E07942][ProgramId=M471010070-service]	기록일시 : 9월 22 07:24:00 WAS실행 쓰레드: 6 사번 : E07942 서비스 : M471010070
Service 시작	9월 22 01:43:00 오후 INFO [[ACTIVE] ExecuteThread: '6' for queue: 'weblogic.kernel.Default (self-tuning)'] - PosBizController.doAction(212) | ServiceName:[M472020090tab01-service] StartTime[Mon Sep 22 13:43:00 KST 2014]	기록일시 : 9월 22 01:43:00 오후 WAS실행 쓰레드: 6 서비스 실행 : PosBizController.doAction 서비스 : M472020090tab01 서비스시작일시 : Mon Sep 22 13:43:00 KST 2014
Service 종료	9월 22 01:45:00 오후 INFO [[ACTIVE] ExecuteThread: '6' for queue: 'weblogic.kernel.Default (self-tuning)'] - PosBizController.doAction(212) | ServiceName:[M472020090tab01-service] EndTime[Mon Sep 22 13:43:00 KST 2014]	Service 시작과 형식 동일
Transaction 시작	9월 22 01:43:00 오후 INFO [[ACTIVE] ExecuteThread: '6' for queue: 'weblogic.kernel.Default (self-tuning)'] - PosDataSourceTransactionManager.startTransaction(122) | Glue: Transaction Start	기록일시 : 9월 22 01:43:00 오후 WAS실행 쓰레드: 6 트랜잭션 실행 : startTransaction
Activity 시작	9월 22 01:43:00 오후 INFO [[ACTIVE] ExecuteThread: '6' for queue: 'weblogic.kernel.Default (self-tuning)'] - PosActivityHandler.runActivity(145) | ActivityName:[M472020090tab01-service][분기] StartTime[Mon Sep 22 13:43:00 KST 2014]	기록일시 : 9월 22 01:43:00 오후 WAS실행 쓰레드: 6 Activity수행 : runActivity 서비스명 : M472020090tab01 Activity명 : 분기 Activity시작일시 : Mon Sep 22 13:43:00 KST 2014
Activity 종료	INFO [[ACTIVE] ExecuteThread: '6' for queue: 'weblogic.kernel.Default (self-tuning)'] - PosActivityHandler.runActivity(188) | ActivityName:[M472020090tab01-service][분기] EndTime[Mon Sep 22 13:43:00 KST 2014] RunTime:[0]	Activity 시작과 형식 동일
쿼리 수행	9월 22 01:43:00 오후DEBUG [[ACTIVE] ExecuteThread: '6' for queue: 'weblogic.kernel.Default (self-tuning)'] - JdbcTemplate.query(526) | Executing SQL query [SELECT /*+ M47 합계정보조회 M472020090tab01_Grid_2.select */ A.COIL_CNT ….]	기록일시 : 9월 22 01:43:00 오후 WAS실행 쓰레드: 6 쿼리수행 : JdbcTemplate.query 쿼리ID: M472020090tab01_Grid_2.select 쿼리문: SELECT /*+ M47 합계정보조회 M472020090tab01_Grid_2.select */ A.COIL_CNT ….
쿼리 Parameter	9월 22 07:24:00 오후 DEBUG [[ACTIVE] ExecuteThread: '6' for queue: 'weblogic.kernel.Default (self-tuning)'] - PosStatementCreatorUtils.setParameterValue(47) | Setting SQL statement parameter value: column index 1, parameter value [A8], value class [java.lang.String], SQL type unknown	기록일시 : 9월 22 07:24:00 오후 WAS실행 쓰레드: 6 파라메터 : setParameterValue 파라메터위치: 1 파라메터값: A8 파라메터 데이터형 : java.lang.String

제 4 장 제안 프레임워크 설계

제 1 절 전체 시스템 개요

본 장에서는 프레임워크를 구현하기 위한 시스템 설계를 한다. 본 논문에서 제안하는 프레임워크는 크게 3부분으로 나누어져 있다. 로그나 이력 데이터를 가져오고 Hive에 적재하기 위해 전처리하는 부분과 Hive저장소, 데이터 추출로 크게 나누어진다. 본 논문에서는 로그수집과 Hive저장소 위주로 설계했다.



<그림 18> 제안 시스템의 전체 개요

제 2 절 세부 시스템 설계

I. 로그 수집

(1) Level 2 로그 수집

3장에서 언급 했듯이 Level 2의 통신 로그는 통신서버에 텍스트 형태로 들어 있거나 운영 DBMS 인터페이스 테이블에 저장되어 있다.

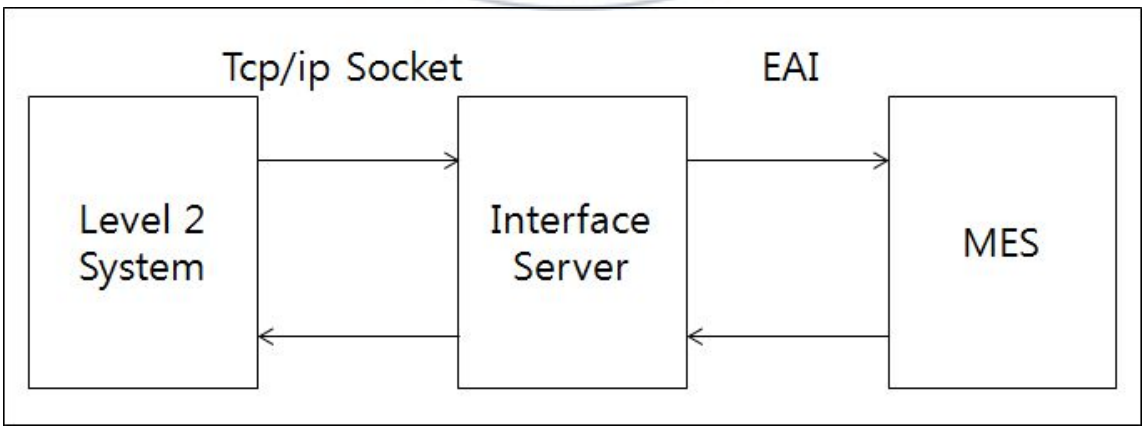
U사의 Level 2통신은 크게 2가지 방법을 사용 한다.

① EAI 사용방식

EAI를 통한 통신은 Level 2시스템과 MES간의 통신을 웹서비스로 직접 호출을 하기 때문에 로그가 서버에 남지 않고 EAI서버에 남는다. WAS를 통해 Level 2로 전송하기 위해 실행되는 서비스나 Level 2로부터 수신된 전문을 처리하기 위한 서비스가 실행될 때 운영 DBMS의 인터페이스 로그 테이블에도 남는다. 실시간성을 고려하면 EAI 통신 시 Flume 같은 스트리밍 도구를 사용하여 받는 것이 좋겠지만 관리하기 까다로운 단점이 있다.

반면 운영 DBMS의 로그 테이블을 사용하면 실시간성은 떨어지지만 SQL 쿼리문을 통해 손쉽게 로그를 획득 할 수도 있고 가공 또한 편한 장점이 있다. 주기적인 실행을 위해 Crontab에 등록하거나 DBMS의 Schedule 기능을 이용해서 로그를 가져 온다면 어느 정도 실시간성은 극복 할 수 있다.

② TCP/IP 소켓



<그림 19> TCP/IP와 EAI 중계 개념

Level 2 시스템이 오래전에 개발되었거나 수정하기 힘든 경우는 중계 서버를 두어 중계 서버가 Level 2 시스템과 MES를 연결한다. 중계 서버는 TCP/IP 소켓으로 Level 2 시스템과 통신하고 MES와는 EAI 통신을 한다.

통신 로그는 중계서버, EAI 서버에 각각 남고 WAS를 통해 서비스가 수행 되면 운영 MES의 로그테이블에도 남는다.

EAI통시방법, TCP/IP소켓 통신방법 두 방식을 모두 고려했을 때 DBMS의 로그테이블에서 가져오는 것이 편의성이나 관리 측면에 더 좋은 방법으로 판단된다.

본 논문에서는 두 방법 중 SQL 쿼리를 통해 획득하는 방법을 채택했다.

(2) 데이터 트랜잭션 이력 수집

생산, 이송/이적 관련 작업이 발생했을 때 DBMS에 작업이력을 반드시 남기게 되어 있다.

생산 이력은 제품을 생산하기 위해 어떠한 공정을 거쳐 왔고 공정간 대기시간과 작업시간을 나타내는 척도가 된다.

이송/이적은 공정 간 원자재, 반제품, 제품의 물류 흐름이 효율적으로 진행되는지 적재 시간이 얼마나 되는지는 아주 중요한 요소이다. 물류 흐름이 효율적으로 진행되지 않을 경우 병목현상이 발생할 가능성이 있고 공장 내 창고의 효율성 저해의 원인이 된다.

동간 이송, 이적 시에 운송 장치들을 통해 실적들이 발생하는데 운송 장치에는 TR(Truck), EC(Elavating car), CC(Coil car), LC(Liner car), OC(Overhead crane), SC(shuttle car) 등이 있다. 실제 운송 장치에서 이루어지는 상차, 하차 실적들은 몇 분에서 몇 십분 소요가 된다. 그러므로 동내 창고에서 재고로 있는 시간에 비하면 무시해도 될 정도다. 또한 운송 장치들은 몇몇 경로가 확실하게

정해진 경우 보다는 동과 동 사이를 자유롭게 움직에게 되어 있는 경우가 더 많아 분석을 더 어렵게 하는 요인이 된다. 그러므로 3장에서 언급했듯이 모든 이송, 이적 이력을 포함할 필요는 없다. 프레임워크에서는 동에서 다른 동으로 이동하는 이송, 이적만을 대상으로 했고 이벤트 또한 상차 이벤트는 제외하고 하차 이벤트만 수집하도록 했다.

데이터 트랜잭션 이력에서 액티비티인 공정과 동의 구분을 위해 생산이력에 대해서는 공정 코드 앞에 'Pr_'을 붙였고 이송, 이적에 대해서는 동 코드 앞에 'Yd_'를 붙여 서로 구분했다.

각각의 테이블에서 조회한 다음 서로의 이벤트를 구분해주기 위해 구분 항목도 추가 했다. 이 3개의 이벤트는 Hive 저장소에 하나의 테이블에 적재 된다.

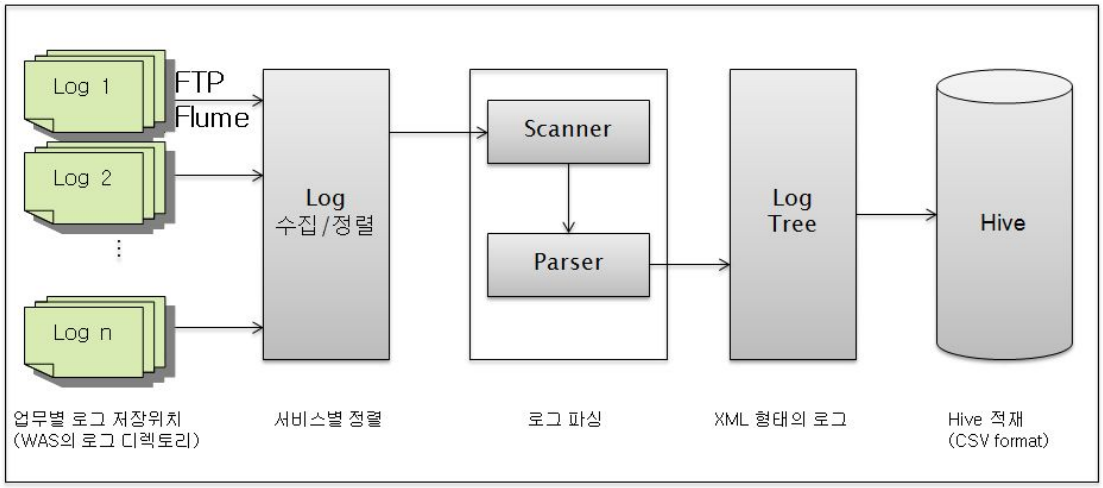
(3) 어플리케이션 실행 데이터 수집

MES의 서비스가 실행되면서 서버에 텍스트 파일로 저장되는 로그를 가공하기 위해서는 3가지 단계를 거친다.

첫 번째, 서버에 저장된 로그 파일은 원시 로그 형태로 가공 시에 필요하지 않은 텍스트를 많이 포함하고 있다. 그래서 처리할 컴퓨터로 가져오기 전에 먼저 전처리를 통해 로그의 량을 줄이는 작업을 한다.

두 번째, 로그를 처리할 컴퓨터로 로그를 가져와야 한다. 로그를 FTP를 통해 가져오도록 구현 했다. FTP를 통해 다운로드 받은 로그는 일자별로 처리할 컴퓨터에 저장된다.

마지막으로 계층형으로 되어 있는 로그를 처리하기 위해서는 스캐너와 파서를 통해 로그를 분석해서 그 결과를 로그 트리로 만든다.



<그림 20> 어플리케이션 로그 처리 전체 프로세스

① 전처리

전처리 모듈은 MES 운영 서버에서 이루어진다. 정규표현식을 사용하여 필요 없는 부분을 제거 하는 역할을 한다.

그림 21은 서버에서 로그를 줄이기 위해 주기적으로 수행되는 Perl 프로그램의 코드이다. @del_rgexp 변수는 문자열의 배열을 저장하기 위한 변수이다. 이 변수에 지울 문자열을 정규표현식 형태로 작성한다.

```
#!/bin/perl
@del_rgexp= (
    'INFO ₩[₩[ACTIVE₩] Execute',
    'DEBUG ₩[₩[ACTIVE₩] Execute',
    '₩[₩[ACTIVE₩] Execute',
    'for queue: ₩'weblogic.kernel.Default ₩(self-tuning₩)₩'₩] ',
    ' PosStatementCreatorUtils.setParameterValue₩(47₩)',
    '₩| Setting SQL statement parameter value: column index',
    ' ₩[[^]]*₩], SQL type unknown',
    'PosJdbcDao.a₩([0-9]*₩) ₩| Executing Query',
    'PosActivityHandler.runActivity₩([0-9]*₩) ₩| ActivityName:',
    'PosValueRouter.doMainActivity₩([0-9]*₩) ₩|',
    'PosDataSourceTransactionManager.commit₩([0-9]*₩) ₩|',
    'PosDataSourceTransactionManager.clearTransactionStatus₩([0-9]*₩) ₩|'
);
$src_folder = "/APP/WAS/UBMWP/LOG/log4j/M47/";
$find_file_name = 'm47.log[0-9][0-9][0-9][0-9][-][0-9][0-9][-][0-9][0-9][-][0-9][0-9][-][0-9][0-9]$';
$cnt = 0;
opendir THISDIR, $src_folder;
@allfiles = grep /$find_file_name/, readdir THISDIR;
closedir THISDIR;

foreach $src_file (@allfiles)
{
    $dest_file = $src_file."b";
    if ($cnt++ >= 5)
    {
        exit;
    }
    print "처리파일$cnt : $dest_file ₩n";

    unless (open READ_FILE, "$src_folder$src_file")
    {
        print STDERR "읽기파일 : $src_folder$src_file 열지 못했습니다.₩n";
    }
    unless (open(SAVE_FILE, ">$src_folder/$dest_file"))
    {
        print STDERR "쓰기파일 : $src_folder/$dest_file 열지 못했습니다.₩n";
    }
    while(<READ_FILE>)
    {
        $str = $_;
        foreach $del_string (@del_rgexp)
        {
            $str =~s/$del_string//;
        }
        print SAVE_FILE $str;
    }
    close(READ_FILE);
    close(SAVE_FILE);
    unlink "$src_folder$src_file";
    $final_file = $src_file."c";
    `mv $src_folder$dest_file $src_folder$final_file`;
}
```

<그림 21> reduce.pl

while문은 파일을 한 줄씩 읽어 $str 에 넣어둔다. foreach문으로 지우기 위해 @del_rgexp에 저장된 문자열 패턴을 읽어 패턴 매칭 후 해당 되는 문자열을 삭제 한다. 이 프로그램으로 다운 받아야 될 로그의 양이 약 50% 줄었다.

② 로그 수집 / 정렬

최근 빅데이터가 대두되면서 로그 수집에 대한 이슈가 점점 커지고 있다. 오픈 소스로도 충분히 좋은 성능을 낼 수 있는 로그 수집기가 꾸준히 개발되고 있다. 아파치 그룹의 flume이 대표적인 예이다.

본 논문에서는 파일 처리를 위주로 하기 때문에 다른 로그 수집기를 사용하지 않고 직접 FTP로 로그를 다운로드 받을 수 있도록 프로그램을 개발 했다.

이중화된 2개의 MES 운영 서버에서 업무 단위로 각각 다른 폴더에 저장된 로그 파일을 주기적으로 가져와야 되기 때문에 서버 정보와 업무 정보를 파일로 지정해서 관리하도록 설계 했다.

```
<?xml version="1.0" encoding="UTF-8"?>
<SERVERS>
   <SERVER ID="PRD1">
          <NAME>운영계1</NAME>
          <IP>210.1.1.101</IP>
          <USERID>userid</USERID>
          <PASSWORD>password</PASSWORD>
          <CHAIN ID="M47">
                <NAME>생산조업</NAME>
                <REMOTE_DIR>/APP/WAS/1/LOG/log4j/M47/</REMOTE_DIR>
                <LOCAL_DIR>D:\Log2Xes\M47\Log\source\1\</LOCAL_DIR>
                <FILE_PATTERN>*.log*</FILE_PATTERN>
          </CHAIN>
              … 중략 …
   </SERVER>
   <SERVER ID="PRD2">
          <NAME>운영계2</NAME>
          <IP>210.1.1.101</IP>
          <USERID>userid</USERID>
          <PASSWORD> password</PASSWORD>
          <CHAIN ID="M47">
                <NAME>생산조업</NAME>
                <REMOTE_DIR>/APP/WAS/2/LOG/log4j/M47/</REMOTE_DIR>
                <LOCAL_DIR>D:\Log2Xes\M47\Log\source\2\</LOCAL_DIR>
                <FILE_PATTERN>*.log*</FILE_PATTERN>
          </CHAIN>
              … 중략 …
   </SERVER>
</SERVERS>
```

<그림 22> ftp.xml의 일부분

FTP 설정 파일은 서버 설정과 각 업무별 로그위치 가져와서 저장할 로컬 디렉토리, 로그 파일 이름을 검색하기 위해 이름 패턴으로 이루어져 있다. 이 파일을 FTP 프로그램에서 읽어서 서버에 접속 후 로그 파일을 다운로드 받는다.

```
public static void loadConfigXml() throws Exception
{
    SAXBuilder builder = new SAXBuilder();
    File xmlFile = new File("FTP.xml");
    FileInputStream   in = new FileInputStream(xmlFile);
    Document doc = builder.build(in);
    Elementroot = doc.getRootElement();
    List<Element> elements = root.getChildren();
    for( Element element : elements)
    {

        System.out.println("Element Name : " + element.getName());
        System.out.println("Element ID : " + element.getAttribute("ID").getValue());
        serverName = element.getAttribute("ID").getValue();
        List<Element> serverElements = element.getChildren();

        for( Element serverElement : serverElements)
        {
           if("IP".equals(serverElement.getName()))
           {
              ip = serverElement.getValue();
           }
           else if("USERID".equals(serverElement.getName()))
           {
              id = serverElement.getValue();
           }
           else if("PASSWORD".equals(serverElement.getName()))
           {
              password = serverElement.getValue();
           }
           else if("CHAIN".equals(serverElement.getName())              )
           {
              System.out.println("Chain ID : " + serverElement.getAttribute("ID").getValue());
              chain =  serverElement.getAttribute("ID").getValue();
              List<Element> chainElements   = serverElement.getChildren();

              for( Element chainElement : chainElements)
              {
                  if("REMOTE_DIR".equals(chainElement.getName()))
                  {
                      remoteDir =chainElement.getValue();
                  }
                  else if("LOCAL_DIR".equals(chainElement.getName()))
                  {
                      localDir = chainElement.getValue();
                  }
                  else if("FILE_PATTERN".equals(chainElement.getName()))
                  {
                      filePattern= chainElement.getValue();
                  }
              }
           }
        }
    }
}
```

<그림 23> LogFtpCient.loadConfig() 함수

그림 23은 ftp.xml 파일을 읽어 설정하는 함수이다.

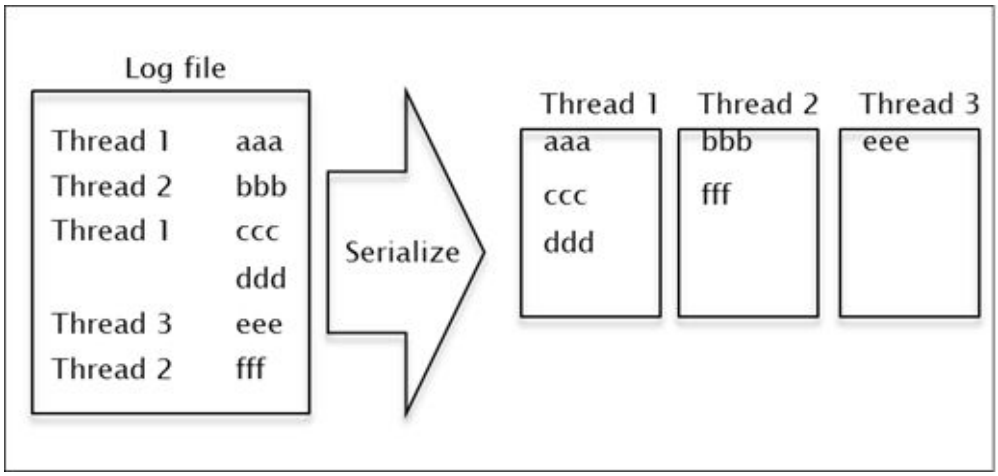
```
public static void ftpRun() throws Exception
{
   LogFtpClient  ftp = new LogFtpClient(ip, port, id, password);
   ftp.connect();
   ftp.login(ftp.id, ftp.password);
   ftp.cd(remoteDir);
   FTPFile[] files  = ftp.ftpList();
   ArrayList<String> localFiles = localList(localDir);
   ArrayList<String> ftpList = new ArrayList<String>();
   // 서버의 로그 파일 리스트
   for(int i =  0; i <files.length;       i ++)
   {
      String fileName= files[i].getName();
      String extension = fileName.substring(fileName.lastIndexOf(".")   + 1);
      long size =files[i].getSize();
      if ( (size > 0)&& (!extension.equalsIgnoreCase("log"))) {
         ftpList.add(fileName);
      }
   }
   //로컬 디렉토리에 파일이 있으면 다시 전송하지 않음
   for(int i =  0; i <localFiles.size(); i ++)
      ftpList.remove(localFiles.get(i));

   for(int i = 0;i < ftpList.size() ; i++)
   {
      String fileName= ftpList.get(i);
      String strYYYYMMDD = getDateDir(fileName);
      String strYYYYMM = strYYYYMMDD.substring(0, 7);
      String localDirFinal = localDir + strYYYYMM + "/" + strYYYYMMDD + "/";
      File dirYYYYMM = new File(localDirFinal);
      if(!dirYYYYMM.exists()) dirYYYYMM.mkdirs();
      ftp.get(fileName, localDirFinal      + fileName);
      System.out.println("내려받기 : " + fileName);
   }
   System.out.println("업데이트에 성공하였습니다.");
   ftp.logout();
   ftp.disconnect();
}
```

<그림 24> LogFtpCient.ftpRun() 함수

그림 24은 설정된 변수를 통해 서버에 접속하여 로그 파일을 다운로드 받는 함수 이다.

MES에서 쓰레드를 통해 동시에 다른 서비스가 여러 개가 수행 된다. 파일에 로그가 출력이 될 때도 시간 순으로 저장되기 때문에 서비스별로 저장이 되지 않는다. 그러므로 로그 파싱을 하기 전에 먼저 쓰레드 별로 정렬하는 작업이 필요하다.



<그림 25> 쓰레드별 로그 정렬의 개념

WAS에서 서비스가 실행되기 위해서는 쓰레드를 할당 받아야 한다. 만약 모든 쓰레드가 작업 중이라면 서비스는 일을 마치는 쓰레드가 발생하기 전까지 대기해야 한다. 동시에 많은 수의 서비스가 수행 되어 서비스 처리가 기록되는 로그 또한 여러 개의 서비스가 뒤섞여 있다. 이를 해결하기 위해 각 쓰레드 단위로 따로 구분하여 버퍼를 두어 쓰레드 단위로 로그를 분리 했다. 쓰레드 단위로 로그를 분리하고 다시 서비스 단위로 분리하면 각각의 서비스를 분리 할 수 있다.

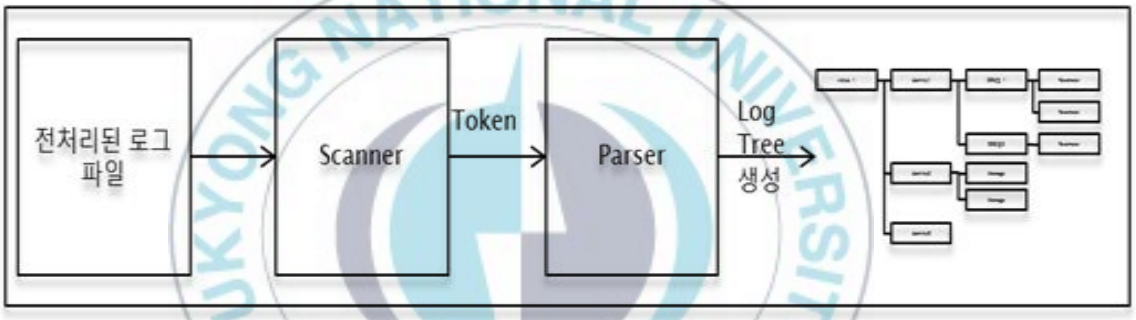
```
while((data = br.readLine()) != null)
{
  Pattern p = Pattern.compile("Thread: '([0-9]*)'")
  Matcher m = p.matcher(data)
  if(m.find())
  {
    threadNo = serverName + "_" + m.group(1)
  }
  if(map.get(threadNo) == null)
  {
    ArrayList<String> ArrStr = new ArrayList<String>()
    map.put(threadNo, ArrStr)
  }
  map.get(threadNo).add(data)
}
```

<그림 26> 쓰레드별 로그 저장 코드

로그를 읽어 쓰레드별로 저장하는 Java Class의 일부이다. 이 클래스는 Map<String, ArrayList> 형태의 Key값과 ArrayList형태의 배열을 map변수를 파라미터로 받아서 서버 번호와 쓰레드 번호를 붙인 Key를 생성하고 파일에서 구분된 쓰레드 배열을 값으로 분류한다. 분류 후 해당 쓰레드 ArrayList에 한 행씩 추가한다.

③ 로그 파싱

로그 파싱을 하기 위해서는 Scanner와 Parser를 구현해야 한다. 로그를 읽어 들여 토큰으로 잘라야 되고 그 토큰의 의미를 파악해서 토큰을 Parser에 입력해서 파싱을 하는데 로그의 성격상 하나의 행이 토큰에 해당 된다.



<그림 27> 로그 파싱 전체 처리 개념도

그림 27는 로그 트리를 생성하기 위해 로그 처리하는 부분의 개념도 이다. Scanner에서는 전처리된 로그 파일이 저장된 List로부터 한 줄씩 읽어서 로그가 무엇인지 판단하여 Token을 발생 시킨다. 다시 Token을 Parser로 넘겨 문법을 인식해서 Log Tree를 생성한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<CONFIG>
<CHAINS>
<CHAIN ID="M47" PATTERN_TYPE="M47"/>
<CHAIN ID="M17" PATTERN_TYPE="M17"/>
... 생략 ...
</CHAINS>
<PATTERNS>
<PATTERN_TYPE ID="M47">
<PATTERN PATTERN_NAME="P_SERVICE_S"
     PATTERN_TEXT=".*PosBizController.doAction.*\[(.*)-service\].*StartTime\[(.*)\]">
</PATTERN>
<PATTERN PATTERN_NAME="P_SERVICE_E"
     PATTERN_TEXT=".*PosBizController.doAction.*\[(.*)-service\].*EndTime\[(.*)\]">
</PATTERN>
<PATTERN PATTERN_NAME="P_SUBSERVICE_S"
     PATTERN_TEXT=".*PosBizController.doSubController.*\[(.*)-service\].*StartTime\[(.*)\]">
</PATTERN>
<PATTERN PATTERN_NAME="P_SUBSERVICE_E"
     PATTERN_TEXT=".*PosBizController.doSubController.*\[(.*)-service\].*EndTime\[(.*)\]">
</PATTERN>
<PATTERN PATTERN_NAME="P_ACTIVITY_S"
     PATTERN_TEXT=".*\[(.*)-service\]\[(.*)\].*StartTime\[(.*)\]" >
</PATTERN>
<PATTERN PATTERN_NAME="P_ACTIVITY_E"
     PATTERN_TEXT=".*\[(.*)-service\]\[(.*)\].*EndTime\[(.*)\]" >
</PATTERN>
<PATTERN PATTERN_NAME="P_QUERY_S"
     PATTERN_TEXT=".*JdbcTemplate.*Executing SQL.*\[(.*)$" >
</PATTERN>
<PATTERN PATTERN_NAME="P_PARAMETER"
     PATTERN_TEXT=".*([0-9][0-9]*), parameter value \[([^\]]*)\]" >
</PATTERN>
<PATTERN PATTERN_NAME="P_MESSAGE"
     PATTERN_TEXT=".*- (.*) [|] (.*)$" >
</PATTERN>
<PATTERN PATTERN_NAME="P_QUERYNAME"
     PATTERN_TEXT="/[*][+].*((M|B)[0-9][0-9].*)[*]/" >
</PATTERN>
<PATTERN PATTERN_NAME="P_USER"
 PATTERN_TEXT=".*PageLogger.doFilter.*userid : ([0-9A-Za-z]*),.*userIp : ([0-9.]*),">
</PATTERN>
... 생략 ...
</PATTERN_TYPE>
... 생략 ...
</PATTERNS>
</CONFIG>
```

<그림 28> LogAnalysis.xml 일부분

로그의 토큰을 인식하기 위한 정규표현식을 저장하고 있는 XML 파일의 일부이다. 업무별로 로그 출력 패턴이 다를 수 있으므로 대표되는 패턴 타입을 정의하고 패턴 타입별로 패턴을 지정 한다.

```
public static int isPatternTypeServiceS(String data)
{
        Pattern p = Pattern.compile(xmlConfig.P_TIME + xmlConfig.P_SS);
        m = p.matcher(data);
        if(m.find())
        {
                return SS;
        }
        return UNKNOWN;

}

public static int isPatternTypeServiceE(String data)
{
        Pattern p = Pattern.compile(xmlConfig.P_TIME + xmlConfig.P_SE);
        m = p.matcher(data);
        if(m.find())
        {
                return SE;
        }
        return UNKNOWN;

    }
    … 중략 …

public static int getPatternType(String data)
{
        int type;

        if((type = isPatternTypeServiceS(data)) != UNKNOWN) return type;
        if((type = isPatternTypeServiceE(data)) != UNKNOWN) return type;
... 중략 ...
       if((type = isPatternTypeUser(data)) != UNKNOWN) return type;

        return UNKNOWN;
}
```

<그림 29> PatternTest.java의 일부분

getPatternType()은 그림 28의 로그 패턴으로 로그를 한줄 씩 입력 받아 해당 문자열을 판단하여 의미를 판단하는 함수이다.

Scanner로 파악된 토큰과 추출 데이터를 이용해서 Parser에서 Log Tree를 만든다. 표 9은 그림17의 로그 구조와 표 8의 내용을 작성한 Log Tree를 만들기 위한 Context Free Grammar 이다.

<표 9>로그 Parsing을 위한 Context Free Grammar

1. LogManager	→ Service LogManager | ε
2. Service	→ SS Activity SE Service | ε
3. Activity	→ AS A-stmt AE Activity | ε
4. A-stmt	→ Query | Message | Sub-Service | ε
5. Query	→ QUERY-STMT Parameter | ε
6. Parameter	→ PR Parameter | ε
7. Message	→ MG Message | ε
8. Sub-Service	→ SSS Activity SSE | ε

Context Free Grammar 는 다음과 같이 풀이 할 수 있다.

1. 쓰레드별로 정렬된 로그는 하나 또는 여러 개의 서비스를 가질 수 있다.
2. 서비스는 서비스시작-액티비티리스트-서비스종료 형태로 되어있다.
3. 액티비티 리스트는 하나 이상의 액티비티로 이루어 졌고 액티비티 시작-액티비티 문장-액티비티 종료 형태로 되어 있다.
4. 액티비티 문장은 쿼리, 메시지, 서브서비스가 올 수 있다.
5. 쿼리는 쿼리 1형식-파라미터, 쿼리 2형식-파라미터 형태로 되어 있다.
6. 파라미터는 하나 이상의 파라미터가 올 수 있다.
7. 메시지는 하나 이상의 메시지가 올 수 있다.
8. 서브 서비스는 서브 서비스 시작-액티비티리스트-서브 서비스 종료 형태로 되어 있다.

Parser를 만들기 위해서 정의된 Context Free Grammar를 변형하여 클래스 단위로 제작 한다.

로그 트리를 만들기 위해 LogManager, Service, Active, Query 클래스를 생성했고 Sub-Service는 Service와 서비스 시작, 서비스 종료 문자열의 일부만 다르고 형태는 동일하기 때문에 Service 클래스에서 처리 하였다. 그 외 A-stmt는

Active 클래스에서 구현했고 Parameter와 Message는 클래스를 만들지 않고 String으로 처리 했다.

```
public  class LogManager extends LogBone implements LogInterface{

 public int logParse(ArrayList<String> al, int alOffset) throws ParseException
 {
  ArrayList<Object> objList = new ArrayList<Object>();
  setObjList(objList);
  for(int i = alOffset; i < al.size();i++)
  {


   switch(pt)
   {
      case SERVICE_S: Matcher m = PatternTest.m;
              Service service = new Service(m.group(3), m.group(4));
              service.setServer(this.getServer());
              service.setChain(this.getChain());
              service.setUserID(this.getUserID());
              service.setIP(this.getIP());

              i = service.logParse(al,  i);
              if(pt == SERVICE_S) service.setRootMode(true);
              objList.add(service);

             break;
      case USER  :  Matcher m = PatternTest.m;
              this.setUserID(m.group(3));
              this.setIP(m.group(4));
              break;

      case SERVICE_E  :
      case ACTIVITY_S :
      case ACTIVITY_E :
      case QUERY_S :
      case QUERY_E :
      case QUERY_SE   :
      case PARAMETER  :
      default       : break;
   }
  }
  return al.size();
 }

 public void generateLogTree(BufferedWriter bw, String id) throws IOException {
  ArrayList<Object> objList = getObjList();
  for(int i = 0; i < objList.size();i++)
  {
   ((JJILogOutput)objList.get(i)).generateXesFile(bw, id + "." + i);
  }
 }
}
```

<그림 30> LogManager.java의 일부분

LogManager 클래스는 쓰레드 단위로 실행이 되며 LogManager 클래스에서 로그 분석이 시작된다. logParse()는 파라미터로 쓰레드별로 정렬된 ArrayList형태의 로그를 입력 받아서 로그 Tree를 생성하는 함수 이다.

```
int pt = PatternTest.getPatternType(al.get(i));
```

위 구문은 <그림 28> LogAnalysis.xml 일부분에 정의된 패턴으로 입력된 문자열을 판단하는 함수 이다. LogManager의 logParser()는 서비스 시작과 유저 정보만 인식하고 나머지 정보들은 다 버린다.

서비스 시작이 발생하면 Service 클래스를 생성하고 Service 클래스의 logParse()를 다시 호출하여 Service 클래스에서 다시 로그 분석을 하도록 한다.

```
public class Service extends LogBone implements LogInterface{
  private boolean rootMode = false;

  public Service(String serviceName, String startTimestamp) throws ParseException {
    super();
    setServiceName(serviceName);
    setStartTimestamp(startTimestamp);
  }
  public int logParse(ArrayList<String> al, int alOffset) throws ParseException
  {
    ArrayList<Object> objList = new ArrayList<Object>();
    setObjList(objList);

    for(int i = alOffset; i < al.size();i++)
    {
      int pt = PatternTest.getPatternType(al.get(i));
      switch(pt)
      {
          case SERVICE_E    : setCompleteTimestamp(PatternTest.m.group(4));
                              return i;
          case SUBSERVICE_E : setCompleteTimestamp(PatternTest.m.group(4));
                              return i;
          case ACTIVITY_S   : Matcher m = PatternTest.m;
                              String activityName = this.getServiceName()+"_"+
                                                    PatternTest.m.group(4).replace(',', '.');
                              Activity activity = new Activity(activityName, this.getServiceName(), m.group(5));
                    ...생략...
                              i = activity.logParse(al,  i);
                              objList.add(activity);
                              break;
          default :           break;
      }
    }
    return al.size();
  }

  public void generateLogTree(BufferedWriter bw, String id) throws IOException {
    ...Log 출력 문장 생략 ...
    ArrayList<Object> objList = getObjList();
    for(int i = 0; i < objList.size();i++)
    {
      ((JJILogOutput) objList.get(i)).generateXesFile(bw, id + "." + i);
    }

  }
}
```

<그림 31> Service.java의 일부분

Service Class의 logParser()는 서비스 종료가 발생하면 호출한 클래스인 LogManager로 돌아가고 서브서비스 종료가 발생하면 호출한 클래스인 Activity 클래스로 돌아간다. 액티비티 시작이 발생하면 새로운 Activity Class의 인스턴스를 생성하고 제어를 새로 생성된 인스턴스로 넘긴다.

```
public class Activity extends LogBone implements LogInterface {
   public Activity(String activityName, String serviceName, String startTimestamp) throws ParseException {
     setServiceName(serviceName);
     setActivityName(activityName);
     this.setStartTimestamp(startTimestamp);
   }
   public int logParse(ArrayList<String> al, int alOffset) throws ParseException
   {
     ArrayList<Object> objList = new ArrayList<Object>();
     setObjList(objList);

     for(int i = alOffset; i < al.size();i++)
     {
          int pt = PatternTest.getPatternType(al.get(i));
          switch(pt)
          {
            case ACTIVITY_E:
                              String activityName = this.getServiceName()+ "_" +
                                     atternTest.m.group(4).replace(',', '.');
                              if(getActivityName().equals(activityName))
                              {
                                 setCompleteTimestamp(PatternTest.m.group(5));
                                 return i;
                              }
                              break;
            case SUBSERVICE_S:
                              Service service = new Service(this.getServiceName(), m.group(4));
                        ...생략...
                              i = service.logParse(al,  i);
                              objList.add(service);
                              break;
            case QUERY_SE  : Query query
                              = new Query(getActivityName(), getServiceName(), m.group(1), m.group(3));
                              i = query.logParse(al,  i);
                              objList.add(query);
                              break;
            case UNKNOWN   :
                              if(PatternTest.isPatternTypeMessage(al.get(i)) == MESSAGE)
                              {
                                 ActivityMessage activityMessage
                                 = new ActivityMessage(getServiceName(), getActivityName(),
                                      PatternTest.m.group(3), PatternTest.m.group(4));
                                 objList.add(activityMessage);
                              }
                              break;
            default : break;
          }
     }
     return al.size();
   }

   public void generateLogTree(BufferedWriter bw, String strCaseId) throws IOException {
     ...Log Tree 출력문장 생략...
     ArrayList<Object> objList = getObjList();
     for(int i = 0; i < objList.size();i++)
     {
          ((JJILogOutput) objList.get(i)).generateCSVFile(bw, strCaseId);
     }
   }
}
```

<그림 32> Activity.java의 일부분

Activity Class의 logParser()는 액티비티 종료가 발생하면 호출한 클래스인 Service로 돌아가고 서브서비스 시작이 발생하면 새로운 Service Class의 인스턴스 생성하고 쿼리 패턴이 발생하면 새로운 Query Class의 인스턴스를 생성하고 제어를 새로 생성된 인스턴스로 넘긴다. 정의된 패턴이 발생하지 않았을 경우는 메시지로 간주하여 처리한다.

Query Class는 다른 클래스와 마찬가지로 패턴 분석 후 쿼리가 여러 줄일 경우 문자열을 붙이는 작업을 하고 파라미터 패턴이 발생할 경우 파라미터 리스트에 저장한다.

<그림 33> 로그 Tree의 예 : DHTMLX의 Tree Grid

로그 Tree는 XML파일과 CSV 두 가지 파일 형식으로 저장이 가능한데 XML 파일로 저장한 경우 DHTMLX의 Tree Grid로 읽어서 로그의 현행업무에서 프로세스 오류 추적이나 분석 시에도 사용 가능하다.

II. Hive 대용량 로그 저장소 설계

수집과 처리를 마친 로그는 Hadoop 기반의 Hive에 저장이 된다. 빠른 분석과 추출을 위해 Hive에서 추출 용이한 형태로 저장한다.

(1) 대용량 로그를 저장하기 위한 데이터 모델

대용량 로그를 저장하기 위한 데이터 모델은 DISCO 같은 프로세스 분석툴에서 쉽게 추출하여 분석할 수 있도록 분석툴에서 요구하는 항목을 필수 항목으로 구성 한다. 또한 각 로그 특성에 맞게 4개의 테이블로 구성했다.

데이터 모델은 프로세스 분석툴에서 필요로 하는 Case ID, Activity, Timestamp와 월별로 파티션을 구성해서 Sliding Window를 구현하게 하는 파티션 Key인 월(YYYYMM)을 필수 항목으로 가진다. 그 외 항목들은 각 로그 종류별로 필요한 항목들을 추가로 가진다.

Level 2 통신 전문 로그 테이블과 데이터 트랜잭션 이력 테이블은 Case ID로 코일ID가 들어간다. 코일ID를 Case ID로 사용할 경우 이점은 공정에서 재료 투입부터 생산과정을 코일별로 추적할 수 있고 더 크게 봐서 데이터 트랜잭션 이력으로 분석할 경우 최초 원재료 치장부터 공정간 이동, 공정들에서의 생산이력, 반제품 치장정보 및 제품생산에 이르는 모든 과정을 추적이 가능하다.

<표 10> Level2 통신 전문 로그

칼럼	내용
YYYYMM	로그 기록월
CaseID	Coil ID
Activity	전문 ID
Timestamp	기록시각
ProcCd	처리공정
Sender	전송 시스템
Receiver	수신 시스템

<표 11> 데이터 트랜잭션 이력

칼럼	내용
YYYYMM	로그 기록월
CaseID	Coil ID
Activity	Unit(Pr_, Yd_)
Timestamp	기록시각
Type	실적 유형
OrgCoilID	원자재 ID

어플리케이션 실행 로그는 서비스나 액티비티 또는 그 하위의 쿼리, 메시지

등 정보 시스템 관점에서 접근한다. Case ID는 숫자형식의 유일키를 생성 하고 Table의 Activity 항목에는 로그가 서비스 항목일 경우 서비스ID 또는 서비스 하위의 Activity일 경우 해당 서비스ID + 로그의 Activity명을 Table의 Activity 항목에 넣는다.

<표 12> 어플리케이션 실행 로그

칼럼	내용
YYYYMM	로그 기록월
CaseID	Sequence Number
Activity	Service / Activity Name
type	Service / Activity / Message / Query / Parameter
StrTimestamp	시작시각
EndTimestamp	종료시각
UserID	Service 시작 전 출력 정보에서 사번 수집
UserIP	Service 시작 전 출력 정보에서 접속 IP 수집

(2) Sliding Window

대용량 로그를 시간 순으로 저장하다 보면 한 테이블에 많은 로그가 누적이 된다. 계속 누적이 될 경우 저장 공간의 선형적인 확장이 지속적으로 필요하게 된다. 계속 누적되는 데이터를 저장하기 위해서 효율적인 저장 방법이 필요하다. 또한 큰 테이블에서 데이터를 추출하기 위해서는 많은 오버헤드를 발생 시킨다. 이를 해결하기 위해 본 논문에서는 슬라이딩 윈도우 개념을 도입했다.

슬라이딩 윈도우는 테이블을 시간 순(월 기준)으로 파티션으로 나누고 적당한 기간을 슬라이딩 윈도우로 지정하여 일정 시간이 지나면 파티션을 아카이빙해서 보관한다.

파티션을 압축하여 저장에 필요한 디스크 공간 사용을 최소화하고 네트워크 부하를 줄여주는 장점이 있지만 로그를 압축하고 압축을 푸는 과정에서 CPU에 많은 부담을 준다. 로그 마다 성격도 다르고 용량도 다르기 때문에 분석에 적당한 크기를 지정하고 그 크기를 슬라이딩 윈도우로 지정해야 한다.

본 논문에서는 현재 월을 제외한 이전 4개월까지를 슬라이딩 윈도우로 지정했다.

Hadoop에서는 파일 입출력 시에 압축파일을 처리 할 수 있는 코덱 사용을 지원한다. 모든 Hadoop 최신 버전은 GZip, BZip2 압축방식과 이런 포맷의 성능을 가속하는 리눅스 라이브러리를 내장하고 있다.[20] Hive는 이런 Hadoop위에서 구동 되므로 문제없이 압축 파일을 입출력 파일로 사용할 수 있다.

```
hive> set io.compression.codecs;
io.compression.codecs=org.apache.hadoop.io.compress.DefaultCodec,
org.apache.hadoop.io.compress.GzipCodec,
org.apache.hadoop.io.compress.BZip2Codec,
org.apache.hadoop.io.compress.SnappyCodec
```

<그림 34> hive에서 사용가능한 코덱 확인 방법

파일 압축은 공간을 절약하는 결과를 줄 수 있으나 원래 압축 파일 형태 그

대로 저장하므로 파일이 분할되지 않는다. 이 경우 병렬처리가 불가하다. 이 문제를 해결하기 위해 Hadoop이 지원하는 시퀀스 파일 포맷으로 저장하면 해결할 수 있다.

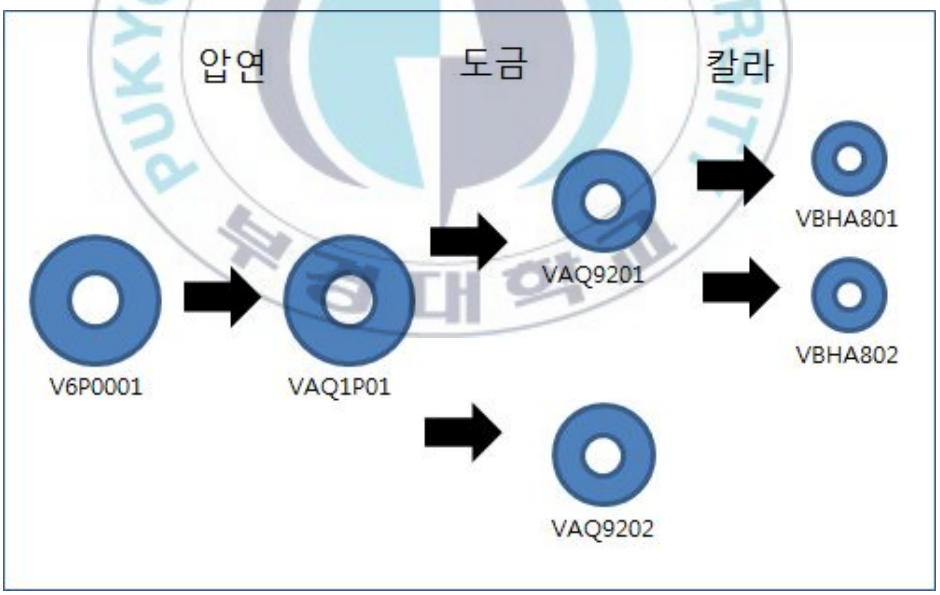
다음은 테이블을 압축하는 코드이다.

```
hive> set hive.exec.compress.output=true;
hive> set mapred.output.compression.codec
      =org.apache.hadoop.io.compress.GzipCodec;
hive> CREATE TABLE logTable1_01_compress
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
    > AS SELECT * FROM logTable1_01;
```

III. 데이터 추출

전체 데이터가 아닌 분석에 필요한 일부분의 데이터를 추출하기 위해서는 추출대상을 가려내기 위한 필터가 있어야 한다. Level 2 통신 전문 로그와 데이터 트랜잭션 이력은 코일ID를 이용해 제품 단위까지 필터가 가능 하다. 어플리케이션 실행 로그는 서비스나 액티비티를 기준으로 추출이 가능하지만 아쉽게도 제품을 가려낼 코일ID 항목 없으므로 제품 단위의 추출은 현재까지는 불가능하다. 하지만 어플리케이션 실행 로그도 서비스 시작 시에 해당하는 코일ID를 특정 포맷에 맞게 출력 하여 추출 가능하다면 제품 단위의 분석이 가능하다. 물론 서비스 시작 부분을 수정해야만 한다. 그 부분은 이 논문 영역 밖이므로 여기서는 제외하고 향후 과제에서 다루기로 한다.

U사의 코일ID는 공정을 거칠 때 마다 새로운 코일ID가 부여 되므로 데이터를 추출하기 위해서는 코일이력이 매우 중요하다.



<그림 35> 코일 분할 과정

최초 투입된 후 코일이 공정을 처리할 때 하나의 코일이 하나 또는 여러 개로 분할 될 수도 있다. 이때 마다 새로 생긴 코일에 대해 이전 이력을 모두 포함한 코일 이력을 만든다.

<표 13> U사 MES 코일이력 테이블 구조

Column	Comment
COIL_ID	코일ID
SEQ_NO	일련번호
PROC_CD	공정코드
FR_COIL_ID	From코일ID
TO_COIL_ID	To코일ID

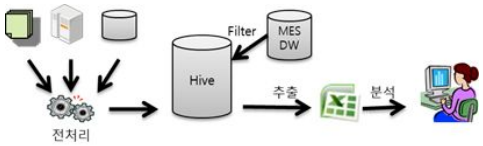
<표 14> 코일 이력 데이터의 예

코일ID	일련번호	공정	From 코일ID	To 코일ID
V6P0001	1	압연	V6P0001	
VAQ1P01	1	압연	V6P0001	VAQ1P01
VAQ1P01	2	도금	VAQ1P01	
VAQ9201	1	압연	V6P0001	VAQ1P01
VAQ9201	2	도금	VAQ1P01	VAQ9201
VAQ9201	3	칼라	VAQ9201	
VBHA801	1	압연	V6P0001	VAQ1P01
VBHA801	2	도금	VAQ1P01	VAQ9201
VBHA801	3	칼라	VAQ9201	VBHA801

표 13은 코일 이력 테이블 구조 이다. 표 14는 그림 35에서 'VBHA801 '의 코일 ID가 어떻게 기록되는지를 보여주는 예시이다. 최초 'V6P0001' 코일이 투입될 경우는 레코드 하나만 생기지만 'VAQ1P01'이 생성될 때 일련번호, 공정, From 코일ID, To 코일ID를 'V6P0001' 로부터 복사해서 코일ID에 자기 코일ID를 넣는다. 다음 도금공정에 투입되면 일련번호 2를 생성한다. 이 과정을 공정처리가 될 때 마다 하여 최종 생성된 코일인 'VBHA801'는 이전 이력인 압연, 도금 공정처리를 포함하여 마지막 처리 공정인 칼라 공정까지 3개의 레코드가 생성 된다. 여기서 이전 이력 코일들은 From 코일ID인 'V6P0001', 'VAQ1P01', 'VAQ9201'이 되고 현재 코일ID인 'VBHA801'를 포함하여 필터를 생성 하는 것이다. 이 필터는 Hive에서 Level2 통신 전문 로그와 데이터 트랜잭션 이력과 조인해서 필요한 추출할 때 사용된다.

IV. 기존 프로세스 마이닝 개발 방법과의 비교

<표 15> 기존 프로세스 마이닝 개발 방법과 제안 프레임워크의 비교

구분	기존방법	제안프레임워크
프로세스	 [21]	
분석 View	· 특별한 구분 없음	· 업무프로세스 단위 · 서비스 단위 · 액티비티 단위
분석 로그형태	· 단순한 형태의 로그 · 같은 포맷 형태의 로그	· 단순한 형태의 로그 · 구문분석이 필요한 복잡한 어플리케이션 로그 · 업무시스템 이력 로그
데이터 Filtering	· 최초 데이터 추출 시 Filtering · 저장된 데이터 항목 Filtering	· 저장된 데이터 항목 Filtering · MES/DW 활용한 특정 제품, 공정, Coil단위의 모든 Filtering 지원
적재/ 추출 방법	· 업무 시스템 내 파일/DBMS내 저장 · 업무 시스템에서 직접 로그 추출/정제/변환 · 분석 시 마다 전체 또는 시간 범위 데이터 추출	· 업무 시스템 로그파일/DBMS 이력을 전처리(정제)하여 Hive에 통합 저장 · 로그 저장소에서 필요한 로그만 선정하여 추출 · 슬라이딩 윈도우 적용으로 최근 로그의 빠른 접근 · 오래된 로그 데이터 압축으로 저장공간 효율적 사용
분석목적	· 업무 프로세스 발견 및 개선 · 어플리케이션 설계 문서 복구	· 업무 프로세스 발견 및 개선 · 어플리케이션 설계 문서 복구 · 정보시스템 성능 개선 · 어플리케이션 오류 추적

제 3 절 프로세스 마이닝 현업 적용 방안

기존 국내 프로세스 마이닝 연구는 주로 사무 프로세스와 서비스 프로세스를 대상으로 진행되어, 제조 분야에 대한 프로세스 분석 및 개선에 대한 연구는 많이 부족한 편이다.

본 절에서는 철강업종을 바탕으로 제조 분야에 대한 프로세스 마이닝 적용방안을 제안한다.

본 논문에서는 MES 실행으로 발생하는 어플리케이션 실행 로그뿐만 아니라 MES내에 발생한 업무 이력 또한 포함 하고 있다. 본 논문을 현업에 적용 시, 프로세스 마이닝 분석을 통해 나타난 문제점을 개선하기 위해 프로세스의 개선 및 확장뿐만 아니라 작업자 교육, 조직구조 개편, 업무규정 변경과 같은 프로세스 외적인 부분까지 개선방안을 제시할 수 있도록 적용해야 한다.

I. 프로세스 마이닝 현업 적용 시 고려사항

프로세스 마이닝을 현업에 적용할 때 고려해야 할 사항은 아래와 같다.

(1) 이벤트 데이터 품질

- 신뢰성 : 실제 일어난 이벤트이며, 속성값은 정확해야 한다.
- 완전성 : 누락되는 것이 없어야 한다.
- 명확성 : 이벤트의 의미가 잘 정의되고 명확해야 한다.
- 정보의 다양성 : 여러 분석 주제를 다루어야 한다.

(2) 프로세스 마이닝 프레임워크 기능성

- 저장성 : 충분한 저장공간이 확보가 되어야 하고 확장이 쉬워야 한다.

- 재가공성 : 저장된 이벤트 데이터를 쉽게 재가공할 수 있어야 한다.
- 데이터 수집의 효율성 : 실시간에 준하는 성능을 확보하기 위해 시스템과 어플리케이션의 성능이 확보되어야 한다.
- 분석 데이터의 효율성 : 프레임워크에 저장된 이벤트 데이터를 필요한 시점에 분석자가 원하는 관점과 범위로 추출이 가능해야 한다.

(3) 조직 및 업무 규정

- 작업자 교육 : 프로세스 마이닝 분석 방법에 대한 체계적이고 전문적인 교육이 필요하다.
- 조직개편 : 프로세스 마이닝 적용 후 지속적인 개선이 가능하도록 전사적 협업 체계 구축과 전담 부서가 있어야 한다.
- 업무규정 : 체계적인 프로세스 개선이 되도록 업무규정이 완비되어야 한다.

II. 프로세스 마이닝 현업 적용 시 부문별 요구사항

(1) 제조 부문

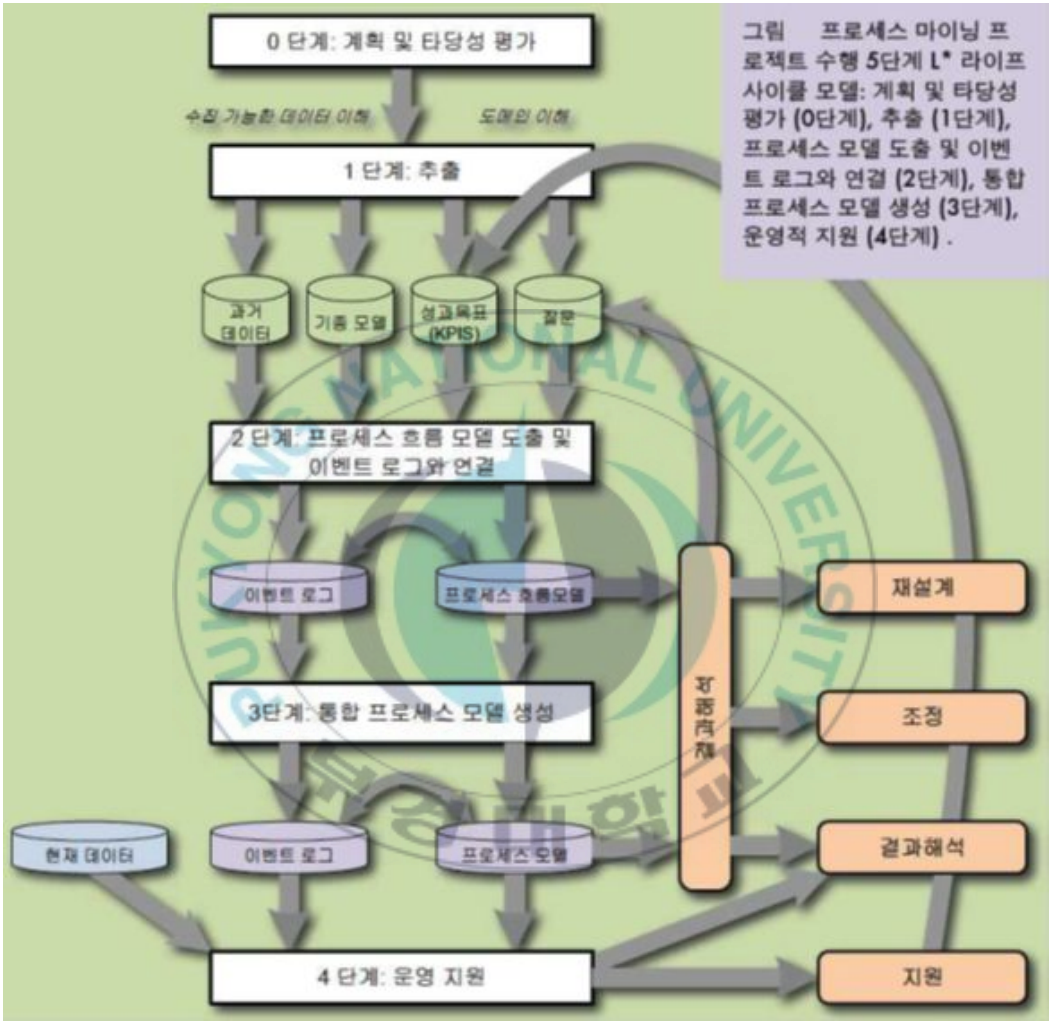
- 작업 프로세스간 유휴 시간 분석 및 최적화
- 물류흐름 분석을 통한 물류 이동 최소화 및 공장 내 창고의 최적화
- 공정 라우팅 분석을 통한 최적의 생산 순서 설정

(2) 기술적 협업 부문

- 병목지점 발견 및 유발 인자 분석
- 불필요한 프로세스 제거
- 설비의 프로세스 최적화

III. 프로세스 마이닝 현업 적용 단계

프로세스 마이닝 현업 적용 단계는 5단계로 이루어진다.[29]



<그림 38> 프로세스 마이닝 프로젝트 수행 5단계

- 0단계 : 프로세스 마이닝 프로젝트 계획 작성 및 타당성 평가
- 1단계 : 프로젝트를 시작한 후에는 시스템, 도메인 전문가, 경영진으로부터 이벤트 데이터와 모델, 목적, 의문사항 등을 도출한다
- 2단계 : 자동화된 프로세스 도출 기법을 활용하여 프로세스 흐름 모델

을 도출한다. 도출된 프로세스 흐름 모델을 바탕으로 기존 프로세스를 재설계하거나 수정 할 수 있다.

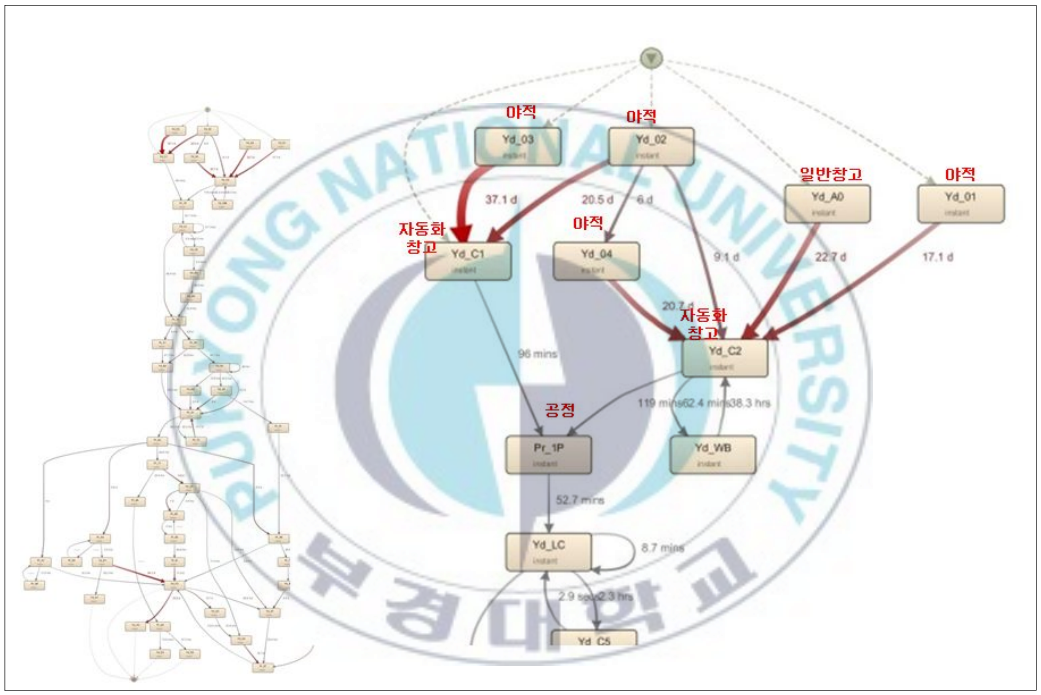
- 3단계 : 프로세스가 비교적 구조적일 때는 3단계에서 프로세스 모델을, 데이터, 시간, 리소스 관점으로 확장할 수 있다. 2단계에서 생성된 이벤트 로그와 모델의 관계를 바탕으로 모델을 확장할 수도 있다.

- 4단계 : 궁극적으로 3단계에서 형성된 모델들을 운영 지원의 목적으로 사용 한다. 과거 이벤트 데이터에서 추출된 지식은 수행 중인 케이스의 정보와 결합 되고, 이는 수행 중인 프로세스에 대한 개입, 예측, 추천에 사용된다. 3단계와 4단계는 프로세스가 충분히 안정적이고 체계적일 때 도달할 수 있다.

제 5 장 로그 분석 수행

제 1 절 업무프로세스 분석

업무 프로세스 분석은 데이터 트랜잭션 이력에서 특정 제품(CCEI, 두께 0.5 이하)에 대해서 필터링 후 수행하였다.

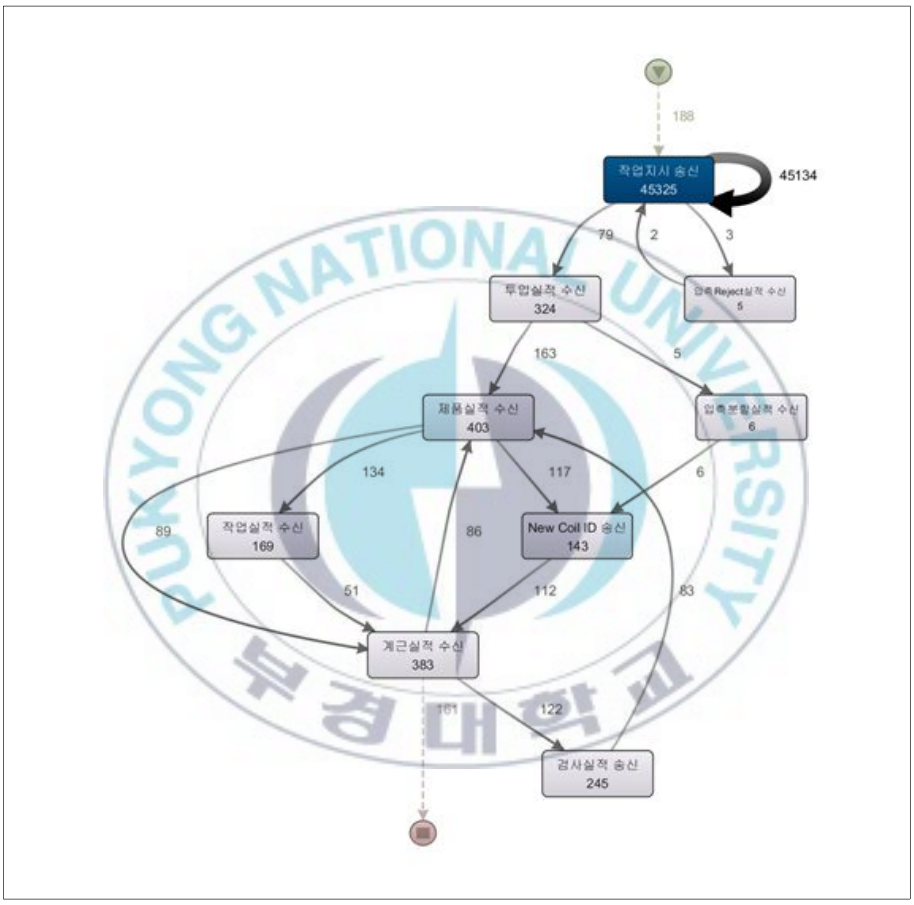


<그림 39> 업무 프로세스 분석 예시

생산을 위해서는 창고에서 Pr_1P공정에 투입이 되는 것이 최초 생산 시작이 된다. 자료를 바탕으로 분석한 결과 자동화 창고에서 Pr_1P로 생산을 위해 가는 경우가 1614건이고 야적 자동화창고에서 Pr_1P로 가는 138건, 일반창고에서 자동화 창고를 거쳐 Pr_1P로 가는 경우가 50건이었다. 야적과 일반창고를 통해서 투입되는 경우가 약 10%가 되는 것을 알 수 있다. 철강의 특성상 야적을 할 경우 비, 눈, 안개 등 습기에 취약한데 야적 평균 체류기간이 20일 이상으로 집계가 됐다.

제 2 절 서비스 단위 분석

1절의 업무 프로세스 분석과 마찬가지로 특정 제품(CCEI, 두께 0.5 이하)에 대해서 칼라공정(A7공정) 작업 이력을 필터링 후 수행하였다.



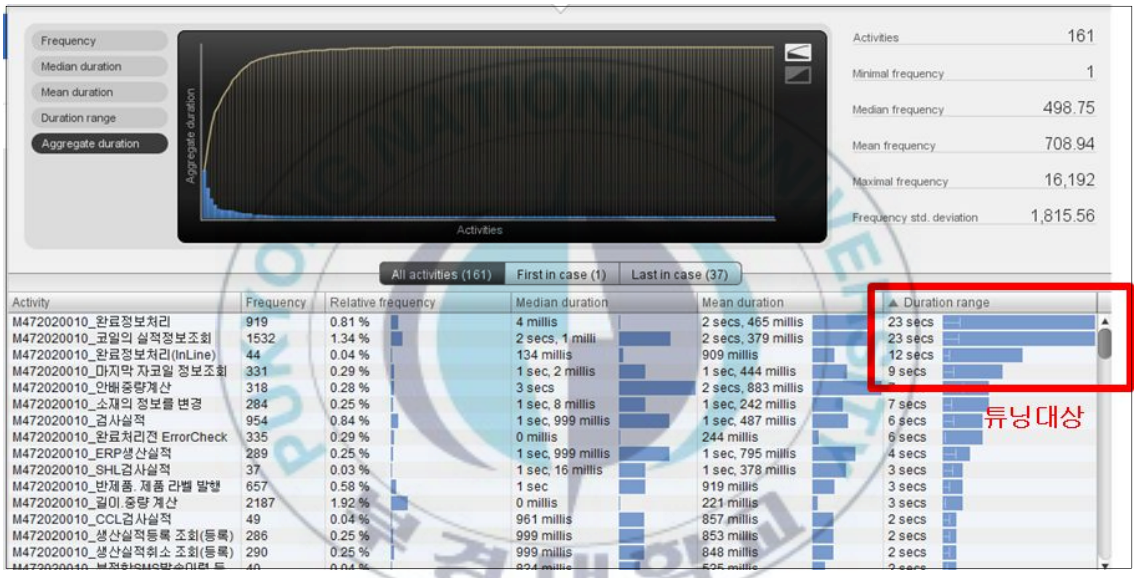
<그림 40> 칼라 공정 프로세스 분석 예시

제 3 절 액티비티 단위 분석

I. 튜닝대상 (검사실적 등록 서비스)

검사실적 등록 서비스만 추출하여 튜닝대상 발굴을 위한 분석을 실시했다.

Duration range가 클수록 시스템 자원을 많이 소모한다. 그림 41에서 Frequency와 Duration range을 참고 하여 튜닝 대상을 찾을 수 있다.

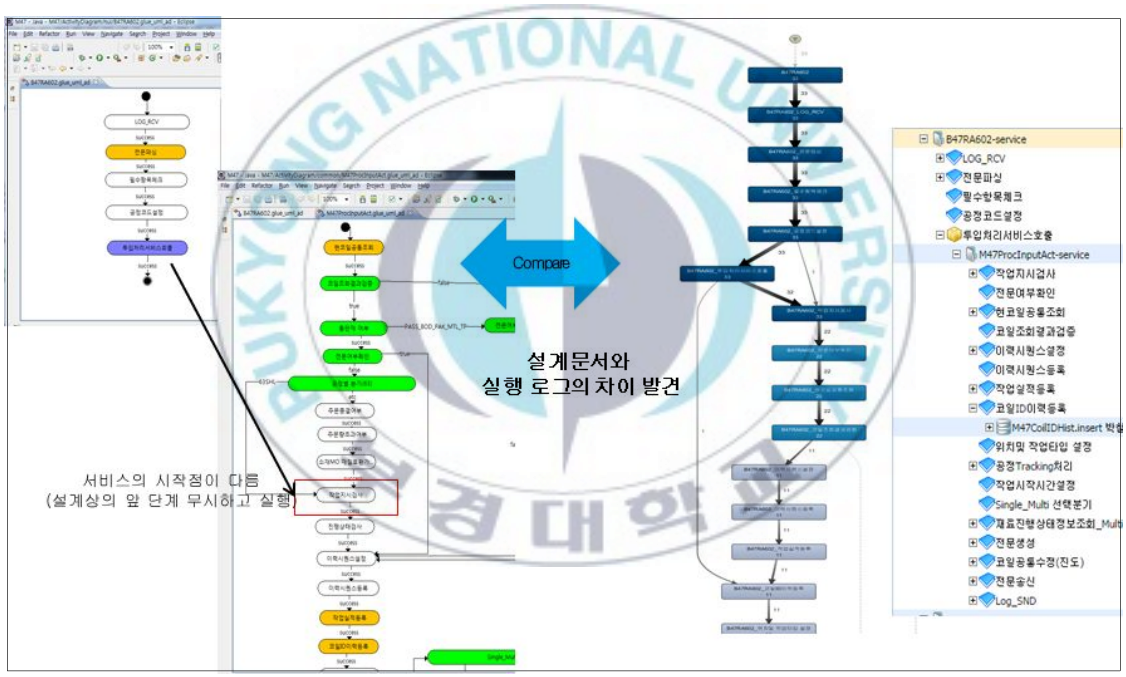


Activity	Frequency	Relative frequency	Median duration	Mean duration	Duration range
M472020010_완료정보처리	919	0.81 %	4 millis	2 secs, 465 millis	23 secs
M472020010_코일의 실적정보조회	1532	1.34 %	2 secs, 1 milli	2 secs, 379 millis	23 secs
M472020010_완료정보처리(InLine)	44	0.04 %	134 millis	909 millis	12 secs
M472020010_마지막 자코일 정보조회	331	0.29 %	1 sec, 2 millis	1 sec, 444 millis	9 secs
M472020010_안배중량계산	318	0.28 %	3 secs	2 secs, 883 millis	
M472020010_소재의 정보를 변경	284	0.25 %	1 sec, 8 millis	1 sec, 242 millis	7 secs
M472020010_검사실적	954	0.84 %	1 sec, 999 millis	1 sec, 487 millis	6 secs
M472020010_완료처리전 ErrorCheck	335	0.29 %	0 millis	244 millis	6 secs
M472020010_ERP생산실적	289	0.25 %	1 sec, 999 millis	1 sec, 795 millis	4 secs
M472020010_SHL검사실적	37	0.03 %	1 sec, 16 millis	1 sec, 378 millis	3 secs
M472020010_반제품, 제품 라벨 발행	657	0.58 %	1 sec	919 millis	3 secs
M472020010_길이,중량 계산	2187	1.92 %	0 millis	221 millis	3 secs
M472020010_CCL검사실적	49	0.04 %	961 millis	857 millis	2 secs
M472020010_생산실적등록 조회(등록)	286	0.25 %	999 millis	853 millis	2 secs
M472020010_생산실적취소 조회(등록)	290	0.25 %	999 millis	848 millis	2 secs

<그림 41> 액티비티 단위 분석(튜닝대상)

II. 설계문서와 실행이력과의 로그 차이 분석

설계문서를 기존 모델이라고 보고 실행로그를 통해 생성된 모델에 대해 적합성 분석을 수행 했다. 그림 42은 기존 모델과 분석을 통해 발견한 프로세스 모델간의 차이를 나타내는 그림이다. 투입처리 프로세스라는 Sub-Service를 수행하는데 설계문서(그림 좌측)와 로그 분석의 결과인 실행이력(그림 오른쪽)의 시작점이 다른 것을 발견하였다. 최초에 설계하여 프로세스 생성 이후에 프로그램 수정이 일어났음을 보여준다.



<그림 42> 설계문서와 실행이력과의 로그 차이

제 6 장 결론 및 향후 연구과제

제 1 절 의미

본 논문에서는 제조업체의 프로세스 마이닝을 위한 프레임워크가 제안 되었다. 다양한 로그를 활용하여 분석 관점별로 분석 사용자별로 정의 하고 분산 파일 시스템에 저장 후 관리하는 방법을 제안하여 기존의 프로세스 절차가 가지고 있는 문제를 개선하기 위한 연구라고 할 수 있다.

본 논문에서 제안한 프레임워크가 가지는 긍정적인 효과는

첫째, 제조 산업인 철강업체의 다양한 로그를 분류하고 활용 방안을 제시하였으며, 분석 관점 및 분석 주체를 세분화 하여 각 상황에 맞게 분석 할 수 있는 방법 로그처리 방법을 을 제시하였다.

둘째, 데이터 생성월을 기준으로 슬라이딩 윈도우 개념을 도입하여 분석이 시급한 최근 데이터에 대해서는 비압축방식으로 저장하고 오래된 데이터에 대해서는 압축 방식으로 저장하여 저장 공간과 CPU부하를 효율적으로 제어 할 수 있는 방법을 제시 하였다.

셋째, MES/DW에 존재하는 비즈니스 데이터를 활용하여 기존의 프로세스 마이닝에 비해 빠른 시간 내에 필요한 데이터를 필터링할 수 있는 방법을 제시하였다.

제 2 절 개선점

로그 수집과 처리에 대해 배치 처리나 FTP를 사용하여 실시간 구현이 되지 않았다. 다음 연구에서는 시스템의 실시간 분석을 위해 실시간 로그 수집을 개선할 예정이다.

몇몇 쿼리나 설정 파일에서 수작업이 현재까지는 많이 필요한데 이는 지속적인 관리를 필요로 한다. 시스템 관리를 최소화하기 위해서는 자동화 도구 사용을 고려하여 개선할 예정이다.

어플리케이션 로그의 경우 앞서 제품 정보로 필터링이 되지 않는다고 밝혔는데 다른 로그 데이터와 마찬가지로 제품의 키가 되는 코일ID를 서비스 수행 시 출력이 되도록 수정이 필요하다. 그러기 위해서는 서비스를 담당하는 부분의 수정이 필요하다.

제 3 절 향후 계획

본 논문은 철강산업 중에서도 냉연공장을 위주로 하였기 때문에 다른 산업과 분석 View라던지 분석 주체가 다를 수 있다. 이는 특정 산업만 적용 가능한 문제점을 발생 시키는데 프레임워크를 좀 더 객관화하고 다른 산업에도 적용 가능 하도록 체계화 하는 연구를 해볼 필요가 있다.

필터 생성을 위한 MES/DW 조회나 추출을 위한 Hive조회를 사람이 일일이 실행 시켜줘야 하는 불편함이 현재 존재한다. 프로세스 마이닝 분석 도구에서 직접 조건을 입력하면 JDBC를 이용해 직접 필터를 생성하고 데이터 추출까지 해주는 플러그인이 있다면 더 유용할 것이다. 그러므로 향후 프로세스 마이닝 분석 도구에서 직접 처리 할 수 있도록 향후 연구가 필요 하다고 본다.

<참 고 문 헌>

[1] 김진숙, "향후 10년 간 IT 기반 10대 비즈니스 트렌드- 맥킨지 보고서를 중심으로"

[2] 한국IDC, "2014년 국내 IT 시장 10대 전망 발표", 2014.

[3] 대한상공회의소, "『빅데이터 활용현황 및 정책과제』연구", 2014.

[4] 가트너, http://www.gartner.com

[5] 정지선, "신가치 창출 엔진, 빅데이터의 새로운 가능성과 대응전략"

[6] 김지숙, "빅데이터 활용과 분석기법 고찰," 2013.

[7] 이재성 and 홍성찬, "기업의 빅데이터 적용방안 연구," 인터넷정보학회논문지, vol. 15, no. 1, pp. 103-112; 한국어, 2014/2 2014.

[8] 서상원, 김재홍, 박윤성, 이준섭, and 명재석, Hadoop & NoSQL, 길벗, 2013.

[9] C. Lam, 이현남, 강택현, 김병곤, 장회수, 원종석, and 김완철, (거침 없이 배우는) 하둡, 지앤선 :, 2012.

[10] A. Holmes, A. Holmes, and 유윤선, 하둡 인 프랙티스, 위키북스, 2013.

[11] http://wiki.apache.org/hadoop/MapReduce

[12] http://nosql-database.org

[13] http://hive.apache.org

[14] 한국인터넷정보학회, "인터넷정보학회지 14(2), 2013.6, 24-31 (8 pages)"

[15] IBM, http://www.ibm.com/developerworks/library/bd-hivewarehouse/

[16] 정재윤, "PROCL : 프로세스 로그 클러스터링 시스템," 한국전자거래학회지, vol. 13, no. 2, pp. 181-194; 한국어, 2008/5 2008.

[17] W.M.P. van der Aalst and A.J.M.M. Weijters. Process Mining: A ResearchAgenda, Computers in Industry, 53(3):231-244, 2004.

[18] 송민석, and 강영식, “프로세스 마이닝 소개 자료”, 2010.

[19] 포스코ICT, https://www.poscoict.co.k

[20] E. Capriolo, D. Wampler, J. Rutherglen, 오세봉, 박영근, 양권국, 우성한, 이종희, 이준섭, 장정호, and 김민우, 하이브 완벽 가이드, 한빛 미디어, 2013.

[21] 강영기, Nita Solehati, and 배준수, "업무 실행 로그데이터를 이용한 프로세스 마이닝 기법 비교 분석," 976-982; 한국어, 2011/5.

[22] 한국방송통신전파진흥원, "빅데이터 분석을 위한 프로세스 마이닝 기술 동향," 2014/3.

[23] van der Aalst, W.M.P., et al. “Process Mining Manifesto,” In F. Daniel, K. Barkaoui, and S. Dustdar (eds.), BPM Workshops (1), LNBIP vol. 99, pp. 169 - 194, Springer, 2011

[24] van der Aalst, W.M.P, Weijters, A.J.M.M., and Maruster., L. "Workflow Mining: Discovering Process Modles from Event logs". IEEE Transcations on Knowledge and Data Engineering, vol. 16, no. 9, pp. 1128-1142, 2003.

[25] Weijters, A.J.M.M., van der Aalst, M.w.p. "Rediscovering Workflow Models from Event Based Data sing Little Thumb". Intergrated Computer-Aided Engineering, vol. 10, no. 2, pp.151-162, 2003.

[26] Giinther, C.W., van der Aalst, W.M.P. "Fuzzy Mining-Adaptive Process Simplification Based on Multi-Perspective Metrics", BPM 2007. LNCS, vol. 4714, pp. 328-323, Springer, Heidelberg, 2007.

[27] van der Aalst, W.M.P. Reijers, H.A. and Song, M. "Discovering Social Networks from Event Logs", Computer Supported Cooperative Work (CSCW), Vol.14, No.6, pp.549-593, December 2005.

[28] Song, M. van der Aalst, W.M.P. "Towards Comprehensive Support for Organizational Mining", Decision Support Systems, Vol.46, No. 1, pp. 300-317, Dec. 2008.

[29] 최상현, "울산광역시 : 산업의 재도약을 위한 프로세서 마이닝(Process Mining) 적용방안에 대한 연구," , 2012.

<표 차례>

<그림 차례>

Development of a Process mining framework in the steel industry

: Focused on MES of the Campany "U"

Jong Ik Jang

Interdisciplinary Program of Management Of Technology, Graduate School, Pukyong National University

Abstract

Many companies have been investing in various information systems such as ERP, MES, APS and SCM for their competitiveness. Over the year, amount of data in business, types of logs created by applications have been rapidly increasing while these information systems have been used, Particularly, although the huge log data has been accrued in the systems depends on business period, the lack of tools that effectively store, extract and analysis it and methodologies led it useless. However, after studies about process mining techniques that effectively analysis log data and find out potential business value released, there is a rising interest on it globally. As the world entered the era of Big Data, tools and methodologies are actively researched for storing and analyzing big data. Therefore, the method that makes the research result effectively used in the field of Process mining should be systematically proposed.

This thesis will categorize type of logs and systematize the process of saving and analyzing from the point of view of Data mining using Hadoop and Hive which are popular big data technologies. It also proposes the method that how to save extracted information into Hive and how to produce research result from Hive through retrieving and refining. In conformance with log data of the steel industry, the proposed methodology in this thesis shows that it can be applied to process mining analysis methodology by analyzing it with DISCO.

Keyword: Process Mining, log Analysis, log analysis methodology, steel industry

감사의 글

끝없이 길게만 느껴지고 힘들게만 느껴지던 논문이 완성되어 제출하게 되었습니다. 처음 시작할 때 과연 내가 할 수 있을까? 이런 의문을 가지고 시작했지만 처음 찾아뵈었을 때 흔쾌히 저를 받아주시고 오랫동안 연락이 안되자 직접 연락까지 하셔서 논문 독려를 해주신 김민수 교수님, 하나부터 열까지 가르침을 받아도 제대로 하지 못해서 송구할 따름입니다. 지면을 통해 감사의 말씀을 전합니다. 그리고 논문심사를 해주신 김영진 교수님, 서원철 교수님 조언 정말 감사합니다. 저 때문에 고생 많이 하신 정현진 조교님 감사드립니다. 휴학으로 적응이 잘 안될 때 같이 공부한 최민욱씨, 박상욱씨 감사합니다. 논문주제선정부터 고민을 얘기 할 때 들어주시고 잘될 수 있도록 조언을 해주신 최우형 형님 감사합니다.

제가 MOT과정에서 공부할 수 있게 해주신 변명섭 고문님, 전종원 본부장님 감사의 말씀을 드립니다. 또 가까이서 격려와 충고로 무사히 마칠 수 있게 도와주신 최진혁 팀장님 감사합니다. 그리고 모든 팀원들에게 감사합니다. 마지막까지 저를 도와준 이돈석 과장, 김정진 대리, 원성옥씨 고마워!!

주말이면 아빠가 학교가서 제대로 놀아주지 못한 우리 두 아들 민석이, 성호 정말 미안하다. 저를 물심양면으로 도와주신 장인어른, 장모님 감사합니다. 저를 낳고 길러주신 어머니, 하늘에 계신 아버지 감사합니다. 마지막으로 내 아내 김옥선씨 정말 사랑하고 감사합니다.