정보통신산업진흥원 부설
nipa SW공학센터

SW아키텍처 참조모델

[빅데이터 기반 시스템의 SW아키텍처 참조모델]

[Version 1.0 20121230]

SW공학센터
SW공학기술팀

SW아키텍처 실무자 포럼 빅데이터 분과
SEC-2012-RM002

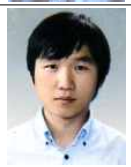
본 문서는 국내 기업의 소프트웨어 품질 및 생산성 향상을 지원하기 위하여 작성되었습니다.

본 문서는
지식경제부 산하
정보통신산업진흥원 부설
SW공학센터
SW아키텍처 실무자 포럼
에서 작성되었습니다.

SW공학센터는 SW제품 생산능력 향상, SW공학기술 산업현장 적용 등을 위해 대학 및 전문 연구기관과 기업 현장을 연결하는 중심 허브가 되어 SW개발 중소기업들에게 전문 컨설팅을 제공하고 있습니다. 이 같은 역할을 충실히 수행하기 위해 산업과 기업의 SW공학기술 관련 요구사항에 전문적이고 신속한 대응을 할 수 있는 핵심 기능 연구와 SW개발 프로젝트의 성공 여부와 문제점을 미리 예측하여 SW품질과 생산성, 제품결함 등을 총체적으로 진단 할 수 있는 SW공학 컨설팅 등도 추진하고 있습니다. 이와 더불어 SW품질과 생산성, 비용 등을 체계적으로 추적 평가할 수 있는 데이터 수집체계도 강화하여 SW기업들이 이를 손쉽게 활용하게 함으로써 전체적인 국가 SW품질을 향상시키는 업무도 수행중입니다.

본 지침에 대한 더 많은 정보와 SW공학에 대한 추가 정보를 얻고 싶다면, SW공학센터 홈페이지(www.software.kr)를 방문하여 주십시오.

담당자 강승준 책임 [ksj@nipa.kr,02-2132-1344]

사진	소속	성명	전자우편
	(주)클라우다인	김병곤	fharenheit@gmail.com
	(주)클라우다인	오재용	obsecure@gmail.com
	KTH	원종석	tedd824@gmail.com
	아이템베이	이용혁	unlogicaldev@gmail.com
	프리랜서	전성욱	allnewangel@gmail.com
	SK Planet	윤재진	spyrogira256@gmail.com
	프리랜서	이승백	hrkensin@gmail.com
	클라우프라	신정훈	jhshin9@gmail.com

목 차

제1장 빅 데이터와 비즈니스 환경

제1절 빅 데이터의 개념

제1장 빅 데이터의 시대 그리고 참조모델의 역할

바야흐로 빅 데이터의 시대이다. 세상이 데이터를 논하고 있고 세상이 인사이트를 논하고 있다. 2011년 하반기부터 '빅 데이터'라는 용어가 세상에 출현한 후 이렇게 빨리 세상에 이슈화되는 것도 보기 드문 일이다(한국이 유별나다). 이제 모든 사람들이 빅 데이터를 이야기하고 있고 빅 데이터를 하지 않으면 기업의 경쟁력이 땅바닥에 곤두박질이라도 칠 것 마냥 떠들고 있으며 마치 빅 데이터가 기업의 핵심 경쟁력이 될 것이라는 전망까지 내놓고 있다.

본 아키텍처 참조 모델은 이러한 세상의 관심과는 별개로 좀 더 현실적인 내용들을 담아보고자 했다. 그래서 본 아키텍처 참조모델을 작성한 현장의 엔지니어들은 다음의 질문을 먼저 던졌다.

○ 빅 데이터 프로젝트는 어떤 프로젝트일까?
○ 빅 데이터 프로젝트를 하고자 할 때 어떤 생각을 하고 있어야 할까?
○ 얼마나 많은 돈이 필요할까?
○ 너무나 많은 오픈소스 중에서 나에게 맞는 오픈소스를 어떻게 선택할 수 있을까?
○ 무엇부터 시작해야할까?
○ 무엇이 정말 중요한가?
○ 실패하지 않는 방법은 무엇일까?
○ 우리에게 주어진 공통의 문제는 무엇인가?
○ 기존의 소프트웨어 아키텍처가 빅 데이터 프로젝트에 적용이 되는가?

빅 데이터는 기존의 애플리케이션 아키텍처와는 많은 차별성을 보인다. 클라우드와 마찬가지로 빅 데이터는 기본적으로 인프라를 먼저 구축해야 한다. 그래서 빅 데이터에서 분석을 논하기 전에 인프라로 활용할 장비를 먼저 고민해야 하고, 그것을 어떻게 구축할 것인지 고민해야 하고, 분석을 위해서 데이터 수집을 어떤 방식으로 무엇을 해야 하는지 먼저 고민해야 한

다(물론 데이터를 어떻게 활용할 것인가도 고민해야겠지만 현실적으로 어떻게 활용할 것인가보다는 수집을 먼저 고민해야 한다). 그래서 빅 데이터에서 분석을 먼저 논하면 어렵다는 것이다). 그렇기 때문에 애플리케이션 아키텍처보다는 시스템 아키텍처가 더 중요한 것이 바로 빅 데이터이다. 본 참조모델에서는 빅 데이터를 처음 접하는 개발자 또는 관리자, 그리고 빅 데이터를 기반으로 프로젝트를 수행하려고 하는 사람들이 시행착오를 최대한 줄이면서 빅 데이터의 기술을 이해하고 빅 데이터의 사업을 이해할 수 있도록 하는 것을 목표로 만들어졌다.

제2장 빅 데이터의 중심은 데이터의 크기와 비용

굉장히 중요한 문제다. 많은 사람들이 빅 데이터라고 하면 아주 큰 데이터라고만 생각한다. 그렇다면 그 큰 크기의 데이터는 어느 정도일까? 10G? 50G? 100G? 1T? 50T? 100T? 300T? 1P? 필자의 경험으로는 정확하게 정의할 수 없다. 단지 현장에서 실제로 구현해 본 현장의 전문가들은 이렇게 정의한다.

"현재 가지고 있는 인프라로 처리할 수 없는 수준의 데이터 크기"

빅 데이터의 "빅"은 조직 내부에서 봤을 때 상대적 의미로 해석해야 한다. 어떤 조직은 데이터의 크기를 200T가 넘는 경우 빅 데이터로 규정하지만 어떤 조직은 데이터의 크기가 50G를 넘는 경우 빅 데이터로 규정하기 때문이다.

그런데 여기에서 중요한 포인트가 하나 있다. "더 이상 처리할 수 없는 수준의 데이터 크기"이다. 만약에 처리할 수 있게 된다면 어떻게 되는 것인가? 저렴한 비용으로 보다 혁신적이면서 새로운 방식을 도입하여 현재 생성되는 크기의 로그 데이터를 요구사항에 충족하도록 처리할 수 있게 된다면 상황은 아주 달라진다. 빅 데이터는 여기에서부터 시작한다. 빅 데이터는 현재 처리할 수 없는 수준의 큰 로그 파일을 처리할 수 있도록 해준다. 이것은 기술이다. 빅 데이터를 이해하기 위해서는 기술을 먼저 이해해야 하는 이유가 바로 여기에 있다.

대용량 데이터를 처리할 수 있는 인프라를 구축하고 데이터를 제대로 활용하는 오픈소스 기반 빅 데이터 조직은 비용적인 측면에서 먼저 접근하는 특징이 있다. 즉, 빅 데이터는 과거에도 현재에도 많은 비용을 투자하면 더 좋은 방식으로 할 수 있다. 문제는 돈이다. 그렇게 하기 위해서 너무나 많은 비용을 투자해야 하며, 매년 높은 수준의 유지보수 비용을 지불해야

한다. 돈이 없는 조직은 아예 접근조차 할 수 없는 방식이다. 하지만 이제 데이터 처리에 투자할 수 없는 작은 기업들조차도 오픈소스인 Apache Hadoop을 값싼 인텔 기반 리눅스 머신에 설치하면 가장 저렴한 비용으로, 더 빠른 성능으로 데이터를 분석하고 처리할 수 있다. 이제부터 Hadoop을 사용하는 조직은 데이터의 크기가 늘어나면 늘어날수록 머신을 추가하기만 하면 된다.

제3장 빅 데이터를 이해하려면 기술을 먼저 이해하라

2003년 Google 엔지니어인 Sanjay Chemawat, Howard Gobioff, Shun-Tak Leung이 'The Google File System'이라는 논문[1]을 발표했다. 이 논문[2]에서 논의하고자 하는 내용은 다음과 같다.

> *We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.*

Google 이 고민했던 바는 엄청나게 큰 데이터를 파일 손실에 대응하면서 고성능으로 처리할 수 있는 파일 시스템을 값싼 하드웨어에서 구현할 수 있도록 하는 파일 시스템을 만드는 것이었다. 그러나 파일 시스템만 만든다면 저장한 파일을 처리할 수 없게 되므로 Google 은 이 파일 시스템의 파일을 처리하기 위한 기술을 정리한 'MapReduce: Simplied Data Processing on Large Clusters'라는 논문[3]을 다시 발표한다. 이 논문에서는 MapReduce를 다음과 같이 설명하고 있다.

---

1) http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/ko//archive/gfs-sosp2003.pdf

2) 번역) http://mittens.springnote.com/pages/74074

3) http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/ko//archive/mapreduce-osdi04.pdf

MapReduce is a programming model and an associated implementation for processing and generating large datasets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper

그로부터 몇 년 후 오픈소스 검색엔진 Lucene의 개발자인 더그 커팅은 Google의 MapReduce와 GFS 기술을 구현한 Hadoop이라는 오픈소스를 세상에 내놓았다. 대용량 데이터를 다루는 조직에서는 이미 오래전부터 Apache Hadoop을 도입하여 데이터를 좀 더 값싼 장비에서 고속으로 처리할 수 있는 환경을 구축하였고 서비스에 적용하고 있었다.

이미 Hadoop을 사용하는 조직은 데이터를 모으고 다루는 일에 능숙했다. 다만 재미난 것은 Hadoop을 이용하던 사람들이 기존의 데이터 분석가가 아니라는 사실이다. 왜 그럴까? 기존의 데이터를 분석하는 사람들은 이미 벤더들이 제공하는 솔루션을 사용하여 데이터를 분석하는 경우가 대부분이나 Hadoop은 그 안에 MapReduce와 HDFS 라는 파일 시스템만 있어서 데이터를 분석하기 위한 도구가 없었으므로 데이터를 분석하고 다루는 등의 작업을 직접 자바 코드로 작성하여 동작시켰다. 그래서 오랫동안 Hadoop을 이용하는 조직은 분석가 중심이 아니라 개발자 중심이다. 개발자가 중심이 되다 보니 기존의 분석가 대비 분석 능력은 현저하게 떨어지지만 분석 그 자체는 상당히 빠르게 적응하고 시도할 수 있게 된다. 그러면서 데이터를 이해하게 되고 분석 기법들을 공부하게 되고 알고리즘을 작성하게 되었고 그것이 서비스로 연결되면서 시너지를 내고, 더욱더 다양한 분석을 하게 된다. 결국 개발자는 기존 분석가 수준은 아니지만 분석가로써 능력을 쌓게 된다. 이 시기에 만약 Hadoop을 데이터 분석가에게 주었다면 분석가는 아무것도 안했을 것이고 여전히 자신이 사용하던 도구를 달라고 외쳤을 것이다.

지금 시장은 빅 데이터를 data scientist, insight, context 등의 용어로 설명하고 있다. 그러나 이러한 용어가 과연 우리에게 현실적인가? 우리에게 어떤 의미가 있는지 이해하고 있는가? 에 답을 줄 수 있는 사람은 거의 없다. 빅 데이터의 성장이 기술이었으나 여전히 분석만 외친다. 그래서 분석을 했던 사람들이 뭐가 다른지 이해하지 못

하고 설명하지 못한다.

제4장 빅 데이터를 특징을 먼저 이해하자

많은 사람들은 빅 데이터하면 analytix, data scientist, insight, context 등의 용어를 먼저 생각한다. 이 용어들은 상당히 고급스럽지만 상당히 추상적이다. 그래서 실제로 구현을 해보기 전에는 어느 누구에게도 설명하기가 어렵다. 빅 데이터 세미나에서 빅 데이터를 논하는 많은 사람들이 실제로 제대로 구현해본 경험이 있었다면 아주 현실적으로 구체적으로 설명을 했겠지만 실제 상황은 그렇지 못하다. 그래서 빅 데이터도 클라우드와 마찬가지로 뜬구름 잡는다고 이야기한다.

하지만 빅 데이터를 이해하려면 가장 먼저 특징을 이해해야 한다. “기존의 프로젝트와 빅 데이터 프로젝트는 어떤 차별화된 점이 있을까?“를 이해하는 것은 빅 데이터를 이해하는데 있어서 매우 중요하다. 다음은 필자가 꼽은 빅 데이터의 특징이다.

○ 기술, 학문, 서비스, 운영기술의 결합
○ 데이터 분석의 대중화
○ 하드웨어와 소프트웨어의 긴밀한 결합
○ 서로 다른 도메인과 기술의 결합
○ 개발과 운영을 동시에
○ 작은 것에서 큰 것으로
○ 시간과의 싸움

빅 데이터는 기술과 기계학습 및 통계라는 학문, 분석 기법들 그리고 운영 기술의 결합이다. 빅 데이터가 쉽게 뿌리를 내리지 못하는 이유는 서로 다른 영역이 합쳐졌기 때문이다. 빅 데이터를 잘 적용한 조직에 있어서 데이터 분석은 이미 개발자들 사이에서 대중화되어 있다고 해도 과언이 아니다. 그 수준은 낮을지언정 대부분의 개발자들이 데이터를 다루고 그 안을 들여다보고 있다. 빅 데이터의 특징이 바로 소프트웨어와 하드웨어의 밀접한 결합이다. 과거에는 비싼 장비를 구매하고 소프트웨어를 구매하면 벤더들이 와서 문제를 해결해주지만 Hadoop은 오픈소스이기에 전부 직접 처리해야 하는 문제가 있으며 운영 중에 발생하는 다양

한 이슈를 직접 해결해야 한다. 또한 좋은 성능을 내기 위해서는 하드웨어에 대한 지식이 필요하고 OS를 적절히 튜닝 해야 한다.

빅 데이터를 한 분야로 보는 시선이 있으나 빅 데이터는 대용량 분산 처리라는 기반 기술일 뿐이다. 그래서 빅 데이터는 어느 도메인이든 적용할 수 있다. 분석 기법도 중요하지만 무엇보다 중요한 것은 해당 도메인의 문제를 해결하는 것이다. 그래서 빅 데이터에서는 분석 기법은 나중 일이고 선행되어야 하는 것이 해당 도메인의 요구사항이고 문제가 된다.

빅 데이터를 구현하기 위해서 여전히 글로벌 표준이 되어 버린 Hadoop을 사용하고 있고 Hadoop은 기본적으로 많은 서버를 필요로 하기 때문에 운영기술이 절대적으로 필요하다. 하지만 대부분의 운영자들이 Hadoop과 같이 분산 기술로 동작하고, 오픈소스이면서 많은 장비로 구성되어 있는 시스템을 싫어한다. 이러한 이유로 대부분의 Hadoop을 관리하는 조직은 운영과 개발을 동시에 한 팀이 진행한다.

빅 데이터는 아주 작은 내부의 문제에서 출발한다. 조직 내부에서 내부 시스템을 잘 운영하기 위한 방법으로 도입하는 경우가 대부분이다. 그렇기 때문에 많은 비용을 투자할 수 없고 작은 문제부터 해결해 나간다. 그렇게 작은 것들을 성공시키면서 학습하게 되고 더 큰 업무에 적용하게 된다. 통상적으로 빅 데이터는 2~3년 정도 개발과 운영 경험을 가져야만 된다. 그래서 작은 것에서 시작해서 큰 것으로 가게 되고 이 기간 동안 시간과 싸움을 벌여야 한다.

이것이 빅 데이터이다. Analytix, data scientist, insight, context가 중요한 것이 아니라 이러한 과정에 집중해야 하고 해당 도메인의 문제를 해결하는데 집중해야 한다. 그래야 빅 데이터가 성공할 수 있다. Hadoop World 2012 New York에서 한 키노트 발표자가 한 말을 우리는 명심해야 한다.

> *"빅 데이터가 도입된다고 인사이트를 얻는 것은 아닙니다. 결국, 인사이트를 알아내는 것은 사람입니다. 빅 데이터는 사람에게 인사이트를 얻을 수 있도록 다양한 데이터 뷰를 보여주는 도구입니다."*

제2절 빅 데이터의 주요 이슈

국내 빅 데이터 비즈니스 환경을 이해하는 것은 매우 어려운 일이다. 유난히 쏠림현상이 심하기 때문이기도 하고 빅 데이터를 경험해본 전문가들이 거의 없으므로 과거의 분석과 차별성을 억지로 찾으려고 한다거나, 대단한 가치가 있는 것으로만 보고 있다. 빅 데이터의 기술적 & 비즈니스 요구사항을 해당 분야의 업무 전문가가 도출하는 것이 우선되어야 한다. 다음은 실제로 빅 데이터와 관련하여 일반인들이 자주 질문하는 내용에 대해서 빅 데이터 전문가(기술)들의 답변으로 설문을 통해 요약한 것이다. 이것이 중요한 이유는 빅 데이터 분야의 특수함 때문이기도 하며 빅 데이터의 시작이 기술이기 때문이다.

주요 이슈	빅 데이터 분야 전문가들의 답변
과연 빅 데이터는 있는가?	• 대부분의 조직은 현재 수준에서 빅 데이터는 없다. • 제조, 통신, 금융(은행/보험/증권 등등), 포탈, 게임 등등의 경우 많은 양의 데이트를 보유하고 있고 데이터 양으로 봐서는 빅 데이터라 할 수 있다. • 다만 빅 데이터는 기존의 데이터를 포함하여 사용자의 사용이력 및 장비에서 발생하는 사용이력까지 과거에 사용하지 않았던 데이터까지 다룬다.
기존 분석과 무슨 차이가 있는가?	• 기존과 같을 수 있고 다르더라도 큰 차이는 없다. • 데이터의 종류가 다양해지고, 그 양이 커져서 분석의 결과가 좀 더 다양하여 차원이 다를 수 있다. • 여전히 분산 환경에서 동작하는 다양한 분석 방법이 개발되어야 하고 기존 분석 방법들도 분산 환경에서 동작하도록 수정되어야 한다. • 분석을 알고리즘으로만 생각해서는 안 된다. 아주 간단한 수식이라도 데이터를 통해 의미 있는 결과를 뽑아낼 수 있다면 그것도 분석이다.
기존 데이터와 어떤 차이가	• 잘 정리된 구조화된 데이터 보다는 빅 데이터는 과거

있는가?	에는 보관이 어렵고 처리하기 곤란한 비정형 데이터 및 사용자와 시스템의 사용 이력을 다룰 수 있다. 이러한 데이터는 데이터가 생성되는 시점과 사용자 또는 장치에 따라서 엄청난 양의 데이터를 생성된다. 당연히 이 경우 빅 데이터 기술을 적용해야 한다.
빅 데이터를 꼭 해야 하는가?	• 빅 데이터는 조직에서 반드시 해야 하는 필요조건은 아니나 • 데이터를 조직을 운영하는데, 서비스를 개선하는데 활용하고자 한다면 도입을 권장한다.
꼭 Hadoop을 써야하는가?	• 충분한 비용을 확보하고 있다면 상용 벤더의 솔루션을 사용해도 무방하다. • 비용이 충분하지 않다면 오픈소스인 Hadoop을 사용하는 것을 권장한다. • 데이터의 규모가 작고, 앞으로도 늘어날 가능성이 없다면 기존 방법을 사용하는 것이 더 생산적으로 얻을 것이 많다. 빅 데이터로 고려해야할 이유가 없다.
현장에서 데이터 과학자가 있는가?	• 없다. • 현장에서는 오히려 데이터 개발자와 업무 전문가 및 해당 분야의 학문적 이해도가 높은 전문가가 필요하다.
빅 데이터 엔지니어가 얼마나 있는가?	• 국내에 거의 없다. • 조직(중요!!)에서 인력 양성에 힘써야 한다. • 빅 데이터는 빅 데이터를 하고자 하는 조직이 기술 내재화를 하지 않으면 성공하기 매우 어렵고 빅 데이터를 수행하고자 하는 조직의 실질적인 요구사항을 도출하지 않으면 실패할 가능성이 매우 높다.
빅 데이터 프로젝트가 실패하는 주요 요인은 무엇인가?	• 여전히 SI 프로젝트로 진행된다. 빅 데이터 프로젝트는 SI 프로젝트와 수행 방법과 접근 방법이 확연하게 차별성을 가진다.

	• 데이터 분석 요건이 명확하지 않아서 목표 시스템 및 서비스가 없다. • 분석을 위한 데이터가 부족하거나 있더라도 데이터의 수준이 많이 떨어진다. • 요구사항이 충분히 개발되지 않고 빅 데이터 용어만 끼워 넣은 프로젝트가 많다. • 기술에 대한 이해도가 전무하다. • 대부분의 프로젝트가 개발 기간이 잘못 산정되어 있다. • Hadoop을 상용 솔루션으로 이해하고 모두 지원되는 것으로 오해한다. • 각종 알고리즘 및 기술(검색 기술, 텍스트 마이닝, 그래프 분석, 추천 등등)을 나열한 RFP가 난무하고 있다.
Hadoop을 사용하는 이유는 무엇인가?	• 저비용, 고효율을 구현할 수 있다. • 빠르게 발전한다. • 글로벌 커뮤니티의 지원을 받을 수 있다.
빅 데이터가 기업에서 중요한 위치를 차지할 것인가?	• 빅 데이터를 제대로 도입하고 내재화를 하는 조직은 분명 차별화된 조직이 될 수 있다. • 하지만 국내 시장은 급하게 성과주의로 풀어나가고 있다. 성과주의는 시행착오를 용납하지 않는다. 빅 데이터는 끊임없는 시행착오를 극복하는 것이 관건이다.
빅 데이터 분석을 하는 경우 개인 정보를 많이 활용하지 않은가?	• 대부분의 빅 데이터 기반기술을 다루고 데이터를 다루는 조직은 굉장히 보안에 민감하여 보안에 철저하다. • 대부분의 조직은 개인 정보를 직접 다루지 않고 장비 또는 사용자의 서비스 사용 이력 정보를 활용한다. • 기업 내부에서 데이터를 활용한다 할지라도 서비스가 다르다면 개인 동의를 얻어야 하므로 기업 내부에서도 데이터를 마음대로 활용할 수 없다. • 개인 정보를 활용하지 않고 분석만으로 어느 정도 유

	추해볼 수 있는 분석 방법들을 연구하고 있다. 이러한 분석 방법은 개인 정보를 활용하지 않고 추정 데이터를 생성한다(예; 그래프 분석 등등)

빅 데이터는 대부분의 조직에서 사업의 방향을 전환할 수 있는 플랫폼이다. 플랫폼은 생태계를 구축하는 것이고 장기적인 안목을 통해 성장한다. 그래서 장기 투자가 되어야 하고 꾸준히 노력하고 확산하도록 해야 한다. 이제는 좀 더 현실적으로 문제를 해결하기 위해서 빅 데이터를 바라볼 때이다.

제3절 빅 데이터 기반 비즈니스 요구사항

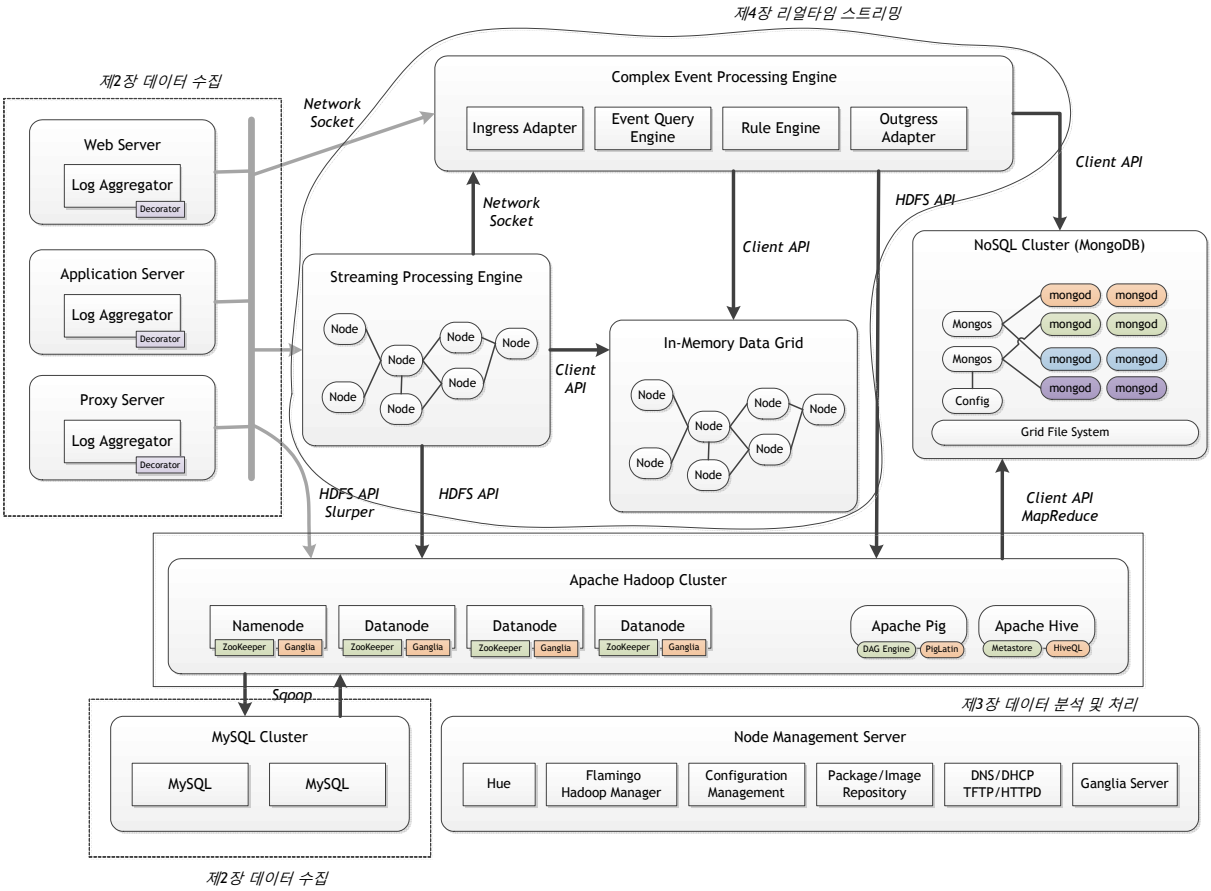
2013년 시작부터 빅 데이터로 시장이 뜨겁다. 공공기관부터 기업까지 빅 데이터라는 주요 이슈를 기반으로 실질적인 비즈니스 요구사항이 시장에 떠돌고 있다. 과연 빅 데이터 시장에서 실제 현장에 있는 고객은 어떤 비즈니스 요구사항을 도출하고 있는지 살펴보았다.

산업분야	요구사항	주요이슈
포탈	스팸 필터링	• 스팸 필터링을 위한 패턴 분석 • 실시간 스팸 필터링을 위한 스트리밍 시스템 구축
	카페 통계	• 대용량의 사용자가 사용한 서비스 사용 이력의 저장 • 서비스 사용 이력에 대한 통계를 생성하는데 소요되는 시간
의료	유전자 분석	• 대용량 유전자 정보 저장 및 처리 • 유전자 염기 서열 구성
교통	자동차 데이터를 이용한 온실가스 배출량 분석	• 자동차 데이터 수집에 대한 법적 이슈 • 대량의 자동차 상태 정보를 수집하고 저장하는 스토리지 • 자동차 상태 정보를 분석하는데 소요되는 시간
금융	현금 인출 및 카드 결제에 따른 실시간 마케팅	• 카드 결제 또는 현금 인출 등과 같은 트랜잭션의 실시간 수집 및 인프라 구축 • 기존 상용 솔루션에서 사용 가능하던 다양한 룰 기반 엔진 적용
통신	부정 사용자 탐지	• 대용량 통화 데이터 및 과금 데이터의 저장 및 처리 • 복잡한 부정 사용자 조건에 부합하는 사용자를

		찾아내기 위해서 개발한 코드의 성능
커머스	사용자의 이용 및 구매 패턴에 따른 실시간 개인화	• 사용자의 구매 패턴을 식별하기 위한 알고리즘의 성능 • 분석한 구매 패턴과 개인화를 위한 시스템 아키텍처 • 실시간 사용자의 사용 이력 수집
클라우드	클라우드 VM의 로그 통합 수집 및 장애 판단	• 장애 판단을 위한 룰 정의 • 장애 판단 조건 변경의 유연성 • 로그 수집을 위한 통합 로깅 표준 • 장애 판단 후처리
	VM의 실시간 모니터링	• 각종 VM의 상태 정보(CPU, 메모리, 프로세스, 디스크 등등) 실시간 수집 및 탐지 • 로그 메시지의 저장 공간
보안	부정 시스템 사용자의 실시간 탐지	• 빠르게 생성되는 대용량 로그 메시지의 실시간 스트리밍 처리 • 분석 패턴의 고도화(대부분 현재 사용하는 기법은 정규 표현식)

제4절 참조 아키텍처

빅 데이터 프로젝트를 수행하면 가장 문제가 되는 것은 시스템 아키텍처를 수립하는 것으로 본 아키텍처 참조모델에서는 다음의 아키텍처 참조모델을 구상하였다.



제5절 요구사항 분석

빅 데이터의 요구사항은 상당히 추상적일 것이라는 일반적은 예상과 다르게 실질적으로 현장에서 발생하는 요구사항은 의외로 기술적이며 업무적으로 단순하다. 다만 상당한 기간 동안 빅 데이터 기반 기술을 활용한 조직은 좀 더 복잡한 기술을 이용하여 복잡한 업무 요구사항을 해결하려고 한다.

1. 요구사항 분석

빅 데이터가 기술적인 문제를 해결하기 위해서 도입을 하는 경우가 대부분이기 때문에 보통 빅 데이터 프로젝트 수행 시 다음의 문제를 해결하고자 하는 요구사항 문서를 작성하게 된다. 업무 요건은 일반적으로 별도의 업무 전문가들이 오랜 시간 동안 고민을 하고 연구하여 작성하기 때문에 여기에서는 별도로 명시하지 않도록 한다.

- 시스템 관련 매트릭스 정보 모니터링
- 노드 확장시 설치 및 설정 자동화
- 최적의 성능을 위한 시스템 및 애플리케이션 파라미터 최적화
- 대용량 로그 처리를 위한 아키텍처
- 통합 및 결함 테스트시 발생할 수 있는 일정 지연 방지를 위한 신속한 개발 방법론
- 품질 향상 방안
- 유지보수 및 효과적인 개발을 위한 데이터 분석 과정 최적화
- 대용량 로그를 분석하기 위한 구현 방법과 기술
- 대용량 로그 파일 증가시 시스템 용량 확보
- 로그 정보 저장 규칙
- 각종 시스템 구성 및 설정에 대한 가이드
- 분석 결과를 타 시스템에 제공하기 위한 연동 방법
- 분석 업무 요구사항에 따른 구현 전략
- 사용자 및 로그 발생 주기가 감소함에 따른 시스템 처리 성능 예측

업무 전문가를 통해 도출된 분석 요구사항을 제외하고 상기 요구사항에 대해서 우선적으로 수행해야할 것은 문제가 될 수 있는 포인트를 도출하는 것이다.

- 시스템 관련 매트릭스 정보 모니터링
  - O 몇 대의 시스템을 모니터링 해야 하는가? 모니터링 대상이 많아지는 경우(예; 20대에서 100대로 증가시) 기존에 모니터링 기법을 사용할 수 없을 수도 있다.
  - O 매트릭스의 종류(OS, Hadoop, JVM 등등)
- 노드 확장시 설치 및 설정 자동화
  - O 어느 수준 까지 자동화를 할 것인가?
  - O 설정 자동화를 하기 위해서 필요한 구성 요소는 무엇인가?
  - O 프로비저닝 기능을 어느 곳에 둘 것인가?
  - O 네트워크 구성이 복잡한가?
- 통합 및 결함 테스트시 발생할 수 있는 일정 지연 방지를 위한 신속한 개발 방법론
  - O 분석 요구사항을 테스트하기 위해서 필요한 테스트 로그 파일이 존재하는가?
  - O 신규 시스템의 경우 로그 파일이 늦게 준비되므로 로그 데이터를 사전에 생성할 수 있는 방법이 존재하는가?
- 품질 향상 방안
  - O MapReduce를 단위 테스트할 수 있는가?
  - O 분석을 위해서 작성한 코드가 독립적으로 테스트가 가능한가?
  - O 코드 커버리지를 사용하여 테스트할 수 있는가?
- 유지보수 및 효과적인 개발을 위한 데이터 분석 과정 최적화
  - O 유지보수가 중요한가? 아니면 처리 성능이 중요한가?
  - O 개발 기간과 테스트 기간이 충분히 확보되어 있는가?
- 대용량 로그 파일 증가시 시스템 용량 확보
  - O 적재한 로그 파일을 타 시스템에게 제공할 필요가 있는가?
  - O 로그의 저장 기간은 얼마나 되는가?
  - O 분석 결과의 유지 기간은 얼마나 되는가?
- 로그 정보 저장 규칙

O 로그 데이터가 생성되는 시간은 언제인가?

O 분석 주기가 어떻게 되는가?

- 분석 결과를 타 시스템에 제공하기 위한 연동 방법

O 최종 분석 결과의 크기가 얼마나 되는가?

O 분석 결과를 사용하는 시스템은 어떤 시스템들이 있는가?

O 분석 결과를 사용하는 시스템이 선호하는 연동 방식은 무엇인가?

O 분석 결과를 데이터베이스에 저장하는 경우 데이터베이스와 Hadoop 노드간 네트워크 연결이 원활한가?

- 분석 업무 요구사항에 따른 구현 전략

O 만족해야 하는 처리 성능은 어느 정도 수준인가?

O Aggregation Function(예; average, sum, min, max)을 어느 정도 사용해야 하는가?

O 분석 코드에서 요구사항 변화에 따라 코드의 변화량이 어느 정도인가?

O 다양한 조건에 따라서 처리해야 하는 경우의 수가 많은가?

O 데이터를 처리하는 과정에서 대규모 정렬이나 조인이 발생할 수 있는 가능성이 높은가?

O 제한된 시간에 수행해야 하는 분석 작업의 개수는 얼마나 되는가?

- 사용자 및 로그 발생 주기가 짧아짐에 따른 시스템 처리 성능 예측

O 최종 목표 사용자의 수

O MapReduce의 입력, 출력, 중간출력의 크기 변화

O Workload의 패턴(IO Bound, CPU Bound)

이제 발생할 수 있는 다양한 질문을 통해서 고객사와 인터뷰 및 조사를 통해 수행 방법을 결정한다.

- 시스템 관련 매트릭스 정보 모니터링

O 모니터링 대상이 많아지는 경우(예; 20대에서 100대로 증가시) 기존에 모니터링 기법을 사용할 수 없을 수 있음. Ganglia 적용 권장.

▷ Hadoop의 Context 설정에 Ganglia 추가 필요

O 모니터링 대상이 적은 경우(20대 이하)

▷ JBoss RHQ와 같은 JMX 및 Agent 기반 모니터링 적용 권장.

▷ Hadoop의 경우 JMX 활성화 필요

- 노드 확장시 설치 및 설정 자동화

O 자동화 수준

▷ OS 수준 : TFTP, PXE 등을 활용한 기술 도입 권장

▷ 애플리케이션 설치 및 설정 수준 : Chef와 같은 설정 서버 도입 권장

▷ 애플리케이션 설정 수준 : 간단한 스크립트 작성 권장

O 프로비저닝 기능의 위치

▷ 별도 서버 구축 : 저사양, 리눅스, DHCP/DNS/TFTP/PXE/ISO 이미지 구축

▷ 타 시스템과 함께 사용 : 리눅스, 네트워크 구성도 확인 필수

O 네트워크 구성의 복잡도

▷ 복잡 : 프로비저닝 서버가 Hadoop과 같은 네트워크에 있는지 확인하고 같은 네트워크에 존재하지 않는다면 멀티캐스트가 가능하도록 구성 권장

- 통합 및 결함 테스트시 발생할 수 있는 일정 지연 방지를 위한 신속한 개발 방법론

O 데이터 검증을 개발 단계에서부터 진행

O 입력 데이터의 가변성 검토(예; null 여부)

O 부동소수점과 같은 검증이 난해한 수치의 경우 검증하고자 하는 소수점 자릿수 결정

- 품질 향상 방법

O MRUnit을 이용한 단위 테스트를 개발단계에서 수행

O MRUnit시 코드 커버리지를 적용

O 다양한 케이스별 로그 데이터 확보

- 유지보수 및 효과적인 개발을 위한 데이터 분석 과정 최적화

O 유지보수가 중요하다면 되도록 코드의 복잡도를 낮추기 위해 분할 정복 시도

▷ MapReduce ETL을 재사용가능하도록 구현하여 분석 과정에 적용

▷ Pig, Hive 등을 분석 과정에 적극 활용

- 예) Multi Column Sorting (Pig), Aggregation Function (Hive)

O 성능이 중요한 경우 하나 이상의 처리를 결합할 수 있는지 판단

▷ 예) 평균값을 계산한 후 분석 결과의 생성 시기를 한 번에 처리

- 대용량 로그 파일 증가시 시스템 용량 확보

O 파일 압축 기법 적용 : Snappy, LZO Compression

- 파일 압축시 추가 라이브러리 설치 및 소스코드 빌드 필요

O 법적 로그 유지 기간이 정해져 있지 않다면 삭제시기를 이른 시간으로 적용

O 분석 결과의 유지 시간을 판단하고 그 기간을 넘어서게 되면 자동 삭제

O 반드시 유지해야 하는 기간이 길고, 물리적인 용량의 한계가 있다면 장비 증설

- 로그 정보 저장 규칙

O 데이터 분석 주기에 따라서 로그 적제 방법을 시간/일/월 등으로 디렉터리 규칙 결정

O 일 단위 분석시 : /YYYY/MM/DD 또는 /YYYYMMDD

O 월 단위 분석시 : /YYYY/MM 또는 /YYYYMM

- 분석 결과를 타 시스템에 제공하기 위한 연동 방법

O 대용량 로그 데이터를 데이터베이스에 적재해야 하는 경우

▷ Sqoop을 이용하여 데이터베이스 직접 INSERT 수행. 데이터베이스의 부하 임계치는 Sqoop의 작업 개수로 지정. 단, 네트워크 연결을 허용해야 하며 데이터베이스는 모든 Hadoop 노드에 권한을 부여해야 함.

▷ Parallel SQL Loader를 이용하여 적제. 단 로그 파일을 SQL Loader가 수행하는 장비에서 접근해야 하므로 FUSE DFS를 이용하여 HDFS를 로컬 파일 시스템으로 마운팅 전략 적용 가능

O 분석 결과를 사용하는 시스템이 다양한 경우

▷ 데이터베이스에 분석 결과 적재

O 분석 결과를 사용하는 시스템간 연동 방식

▷ 신규 설치를 상대 시스템이 권장하지 않는 경우 FTP 사용 권장

▷ FTP 이용시 파일명 및 디렉터리 규칙 결정

▷ FTP 처리 코드의 관리 주체 결정 및 장애 대응 방안 수립

▷ 업로드 중간에 연동 시스템이 파일 처리를 않도록 하기 위해서 처리중임을 표시할 수 있는 네이밍 규칙 결정 (예; 업로드 중 → .txt.idx / 업로드 완료 .txt)

- 분석 업무 요구사항에 따른 구현 전략

O 높은 처리 성능이 필요한 경우 Hive → MapReduce → Pig 순서대로 기능 제공 여부 검토

O Aggregation Function(예; average, sum, min, max)을 빈번하게 사용하는 경우 Hive 권장

O 요구사항 변경에 따른 코드 추가 및 변경이 잦은 경우 MapReduce 권장

O 결과값을 도출하기 위해서 복잡한 조건이 결합되는 경우 MapReduce 권장

O 멀티 칼럼 조인 또는 정렬이 발생하는 경우 Pig 권장

O 알고리즘 개발시 MapReduce 권장

O 단순 Aggregation Function이 필요한 경우 Hive Function 개발 권장

- 사용자 및 로그 발생 주기가 짧아짐에 따른 시스템 처리 성능 예측

O 목표 사용자에 따른 예상 로그 데이터의 크기를 예측

O 분석 업무의 데이터 처리 패턴 식별

O 분석 업무와 유사한 처리 패턴에 대한 벤치마킹 수행

▷ DFS IO (I/O Bound)

▷ MRBench (CPU Bound)

▷ Wordcount (CPU Bound)

▷ Terasort, Teragen (CPU, I/O Bound)

O 데이터 증가에 따른 I/O 및 CPU 사용량 확인

2. 담당자별 역할

빅 데이터 프로젝트는 보통 다음의 담당자가 관여하지만 조직에 따라서 하나 이상의 역할을 한 명의 담당자가 처리하는 경우 많다. 다음의 담당자 역할은 빅 데이터 영역에 특화된 담당자 및 역할을 설명한 것이므로 일반 프로젝트와 매우 다를 수 있다.

담당자	역할
업무 전문가	• 특정 도메인에 대한 전문 지식을 제공한다. • 해당 도메인의 중요한 요구사항을 도출한다. • 일반적으로 고객사의 구성원으로 활동하는 경우가 많다.
데이터 분석가	• 업무 전문가의 요구사항을 분석하고 분석 요구사항을 도출한다. • 개발을 위한 입력 데이터를 제공한다. • 데이터 검증을 위한 테스트 데이터를 제공한다. • 개발자가 데이터 분석 프로세스를 구현하기 위해서 필요한 상세 설계 문서를 작성한다.
개발자	• 데이터 분석가의 설계 문서를 참고하여 구현한다. • 입력 데이터를 이용하여 데이터 분석 과정을 수행하고 검증 데이터로 처리 결과를 검증한다. • 분석 업무 프로세스의 성능 시험 및 튜닝을 진행한다.
아키텍트	• 업무 전문가, 데이터 분석가, 개발자의 커뮤니케이션을 조율한다. • 업무 요구사항 및 분석가의 설계 문서를 기반을 MapReduce, Pig, Hive 등의 적합한 기술을 제시한다. 필요시 적당한 성능 시험을 수행한다. • 개발자의 개발 환경에 대한 표준안을 마련한다. • 물리/논리 시스템 구성도를 도출한다. • 성능 및 확장성과 같은 비기능적 요구사항을 도출하고 문제를

	해결하기 위한 적절한 해결책을 제시한다. • 시스템 간 데이터 연동 방식을 결정한다(예; FTP, Sqoop, NFS, FUSE 등등). • 분석 업무 프로세스의 배치 스케줄링 계획을 수립한다. • 데이터의 검증 전략을 수립한다. • 기술 교육을 진행한다. • 각종 오픈소스 배포판의 배포 전략을 수립한다. • Hadoop Cluster의 노드 관리 및 프로비저닝 전략을 수립한다.

3. 입력/출력물 정의

빅 데이터 프로젝트는 데이터를 중심으로 프로젝트가 전개되기 때문에 타 프로젝트 대비 산출물 목록에서 집중해야 하는 특별한 산출물이 존재한다. 기본 산출물을 제외하고 빅 데이터 프로젝트의 관련 산출물에서 다음의 내용이 포함되어 있는지 확인하도록 한다.

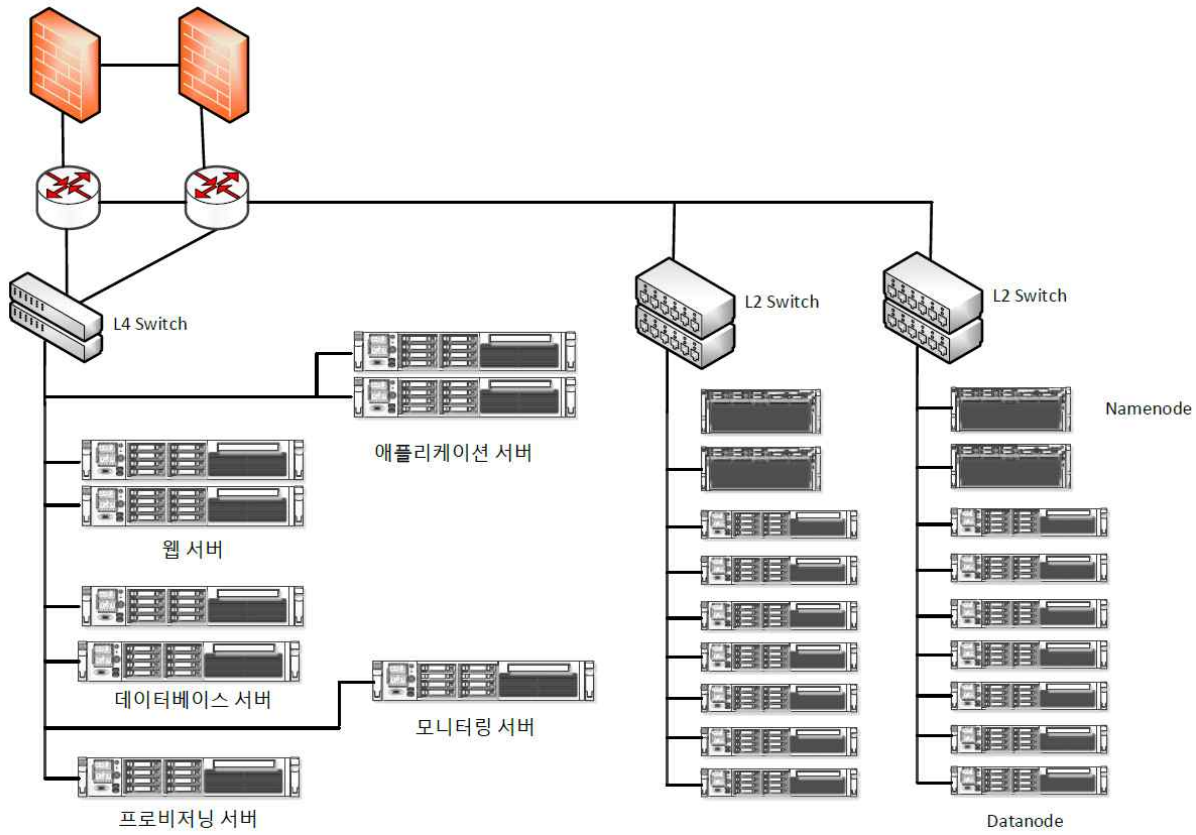
입/출력물	설명
데이터 정의서	• 로그 데이터의 칼럼 정보를 포함한 메타 데이터(데이터의 컬럼명, 필수여부, NULL 여부, 자료형, 자릿수 등등) • 데이터의 생성 주기 • 데이터의 소멸 주기 • 데이터 처리 주기 • 데이터 생성시 예상 크기 • 분석 수식
설계 문서	• 분석 업무 프로세스의 데이터 플로우 정의 • 분석 업무 프로세스의 입출력 데이터 정의 • 분석 업무 프로세스의 중간 데이터 정의 • 파라미터 정의

	• 클래스 정의 • 비기능적 요구사항 정의(처리 임계 시간, 데이터 정확도, 유지 보수성)
개발 가이드	• MapReduce 개발 템플릿 및 프로그래밍 가이드 • MapReduce 파라미터 설정 가이드 • MapReduce 개발 가이드 • 단위 테스트 작성 가이드
환경설정 가이드	• Apache Hadoop, Pig, Hive 설정 정보 • MySQL 설정 정보 • Ganglia 설정 정보(모니터링 구축시)
설치 가이드	• 운영체제 설치 및 패키지 리스트 • Apache Hadoop, Pig, Hive, NoSQL 설치 • HDD 파티셔닝 및 파일 시스템 구성
소스코드	• MapReduce 소스코드(Driver, Mapper, Reducer, Combiner, Partitioner, Input Format, Output Format, Record Reader 등등) • Pig Function • Hive Function • Hive QL • Pig Latin Script
용량 산정 결과서	• 분석 업무 프로세스 및 데이터 처리 프로세스의 처리 용량별 성능 결과 • Datanode 대수, 입력/출력 데이터의 Read/Writer에 따른 처리 성능 결과 • 적용한 workload 패턴

제6절 시스템 구조

1. 물리 시스템 구조

빅 데이터 기반 시스템의 일반적인 시스템 구조는 다음과 같다. 물리 시스템 구조로써 서비스 제공을 위한 웹 서버, 애플리케이션 서버, 데이터베이스 서버가 있으며 시스템 자동 설치 및 설정을 관리하는 프로비저닝 서버, 그리고 각 노드의 상태 정보를 모니터링 하는 모니터링 서버가 있다. 그 외 Apache Hadoop을 구현하기 위한 Namenode와 Datanode를 다수 준비하여 하나 이상의 클러스터를 구축한다. 대부분의 경우 Hadoop Cluster는 Rack 단위로 네트워크 스위치를 준비한다.



2. 소프트웨어 구성요소

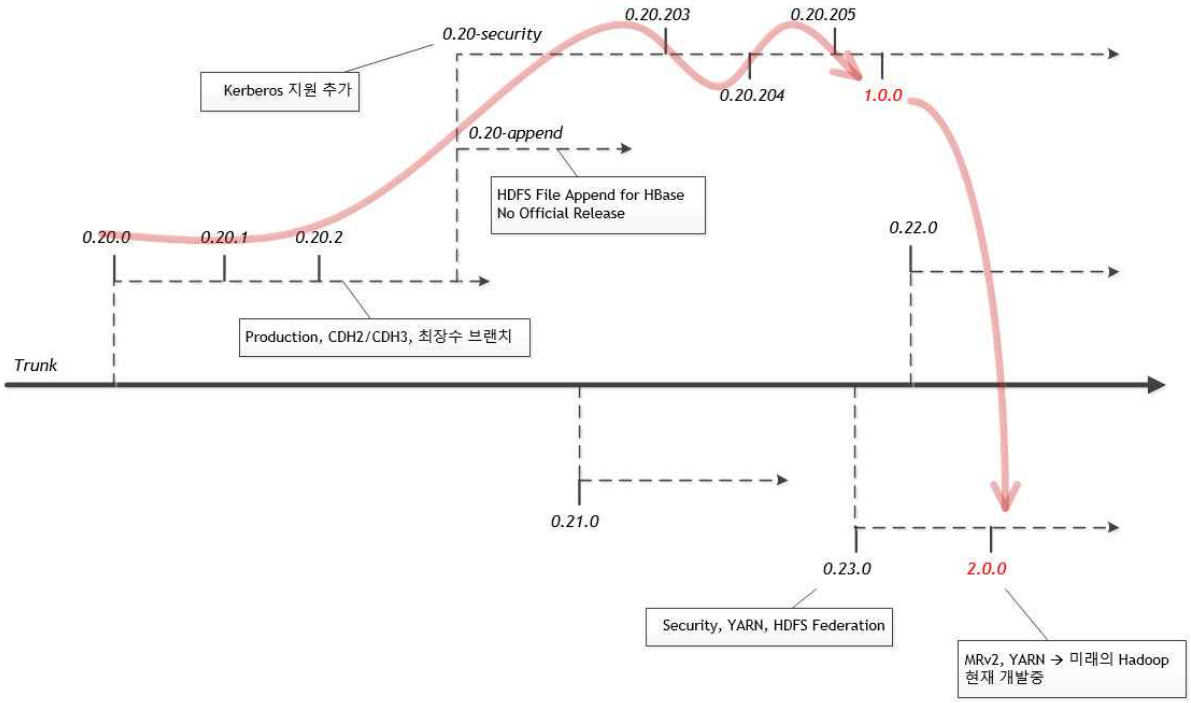
Apache Hadoop을 기반으로 빅 데이터를 구현하는 경우 버전의 호환성이 매우 중요하다. 따라서 각 장비별로 설치하고자 하는 경우 다음과 같이 각 노드별 설치 버전과 설치해야할

항목을 결정하는 것은 매우 중요하다 할 수 있다.

구분	설치 컴포넌트	버전	설치 노드					
			프로비저닝	Database	Name Node	Job Tracker/ Secondary	Data Node	로그 수집기
OS (RHEL 6)	Snappy Compression				O	O	O	O
	iperf		O		O	O	O	O
	Ganglia	3.3.0	O		O	O	O	
	MySQL	5	O		O	O	O	
	DNS(Bind)	9.8	O					
	DHCP	4.1.x	O					
	HTTP	2.2.15	O					
	TFTP	0.49	O					
	NTP	4.2	O					
	SSH	5.3	O					
	rsync	3.0.6	O					
Apache Hadoop EcoSystem	Apache Hadoop	1.1.1	O		O	O	O	O
	Apache Pig	0.10.0	O		O	O	O	O
	Apache Hive	0.9.0	O		O	O	O	O
	Sqoop	1.4.2	O		O	O	O	
	OraHive	1.1.0	O		O	O	O	
기타	Oracle JDBC Driver	11.2.0.3	O		O	O	O	O
	MySQL JDBC Driver	5.1.21	O		O	O	O	O
Batch Job	MapReduce Job		O		O			
	Sqoop Job		O		O			
	OraHive Job		O		O			

3. Apache Hadoop 배포판 선택

Apache Hadoop 위에서 동작하는 Hive, Pig, Mahout 등의 오픈소스는 앞서 설명한바와 같이 버전간 호환성이 매우 중요한데 호환성을 이해하려면 가장 먼저 Apache Hadoop의 변화한 상태를 이해할 필요가 있다. 다음은 Apache Hadoop의 계보로써 계보에 따라 현 단계에서 적용해야 하는 버전은 1.x 버전이며 차후 적용해야 할 버전은 2.x가 된다.

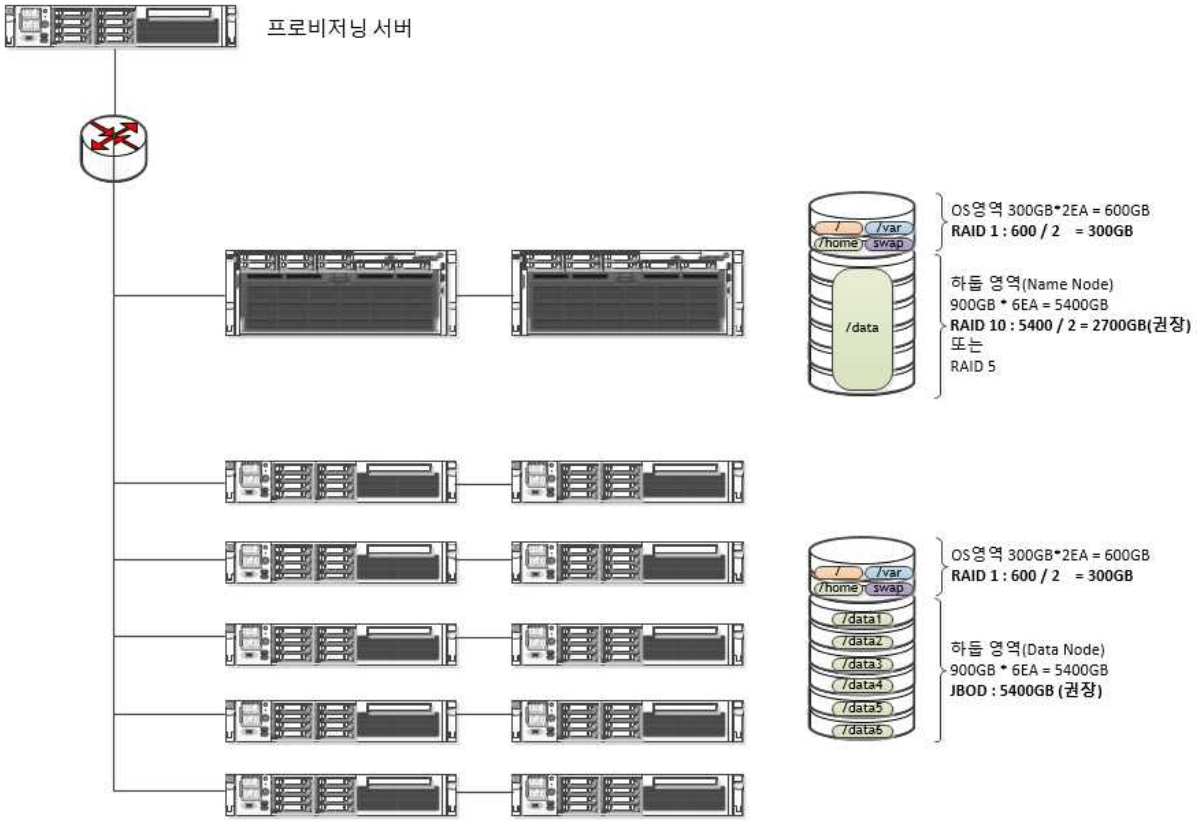


각각의 버전에 따라서 다양한 기능이 추가되고 비공식 릴리즈가 이루어졌으므로 다양한 배포판이 존재하게 된다. 이것은 결국 사용자가 적절한 버전을 선택하는 것이 매우 어렵다는 것이다. 중요한 기능을 중심으로 다음과 같이 버전별 지원 기능을 도표로 표시해보면 Cloudera 배포판과 Apache 배포판은 기능상 차이가 있음을 알 수 있다. 특히 CDH4의 경우 Hadoop 1.x와 Hadoop 2.x의 모든 기능을 결합한 버전으로 이 버전을 선택하는 경우 lock-in 될 가능성이 매우 높을 것으로 예상된다.

Feature	0.20	0.21	0.22	0.23	1.0 [1]	2.0	CDH3	CDH4 [2]
Production	O				O		O	O
HDFS Append		O	O	O	O	O	O	O
Kerberos		O (HDFS)	O	O	O	O	O	O
HDFS symlinks		O	O	O		O		◇
YARN(MRv2)				O		O		◇
MRv1	O	O	O		O		O	O
Namenode Federation				O		O		◇
Namenode HA				O		O		◇

4. 하드 디스크 파티셔닝

Apache Hadoop의 경우 파일의 메타 데이터를 관리하는 Namenode와 실제 파일을 저장하고 처리하는 Datanode로 구성할 수 있다. 다음은 최적화된 하드 디스크 파티셔닝을 예시로 표현한 것으로 Namenode의 경우 안정성을 이유로 OS 영역과 메타 데이터를 저장하는 공간에 RAID를 구성하고 복구 수준에 따라서 level을 다르게 적용한다. 이와 다르게 Datanode는 고성능 파일 시스템을 구현해야 하므로 RAID를 적용하지 않고 JBOD를 적용해야 한다. RAID를 적용하는 경우 성능 저하가 발생한다.



5. 시스템 모니터링

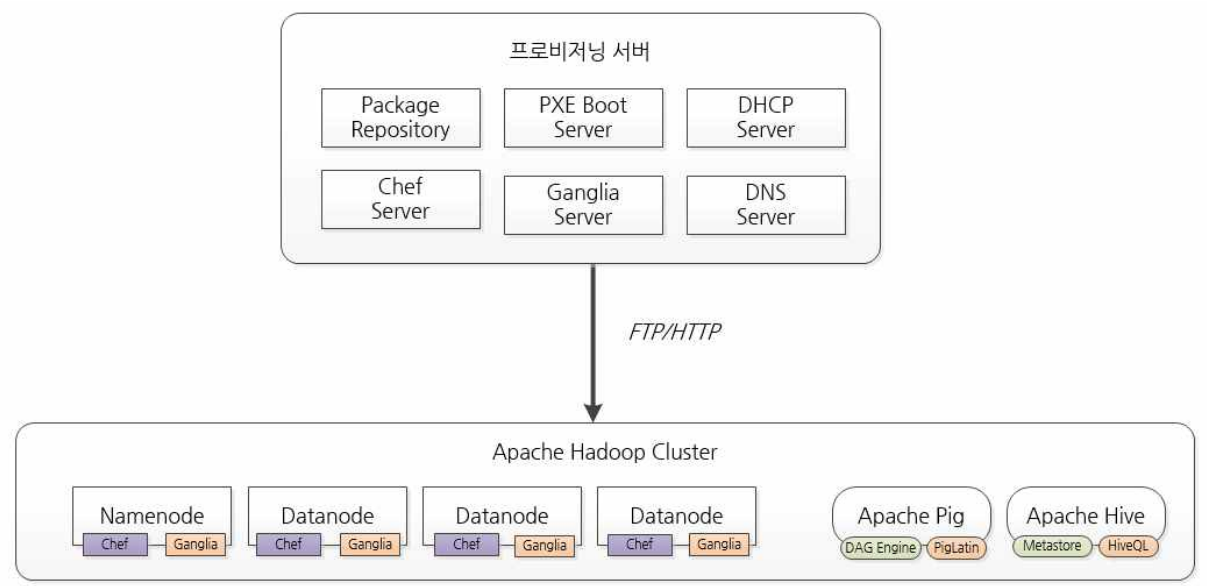
시스템 모니터링을 적용하고자 하는 경우는 클러스터의 크기에 따라서 서로 다른 선택을 할 수 있으며 일반적으로 Hadoop의 경우 대규모 클러스터에도 적용가능은 Ganglia를 사용한다. 다만 Ganglia를 사용하기 위해서는 각 노드에 Ganglia 데몬이 설치되어 있어야 하며 Hadoop에 Ganglia와 관련한 설정 정보를 추가해야 한다.

Ganglia를 통해 수집해야 하는 최소 정보는 다음과 같다.

- OS의 메모리, CPU, 네트워크 사용률
- Datanode의 디스크 사용률
- Datanode의 Map/Reduce Task 개수
- Datanode의 Thread 개수
- Datanode의 네트워크 트래픽
- Datanode의 JVM Heap
- Datanode의 JVM GC

6. 프로비저닝

많은 수의 노드를 관리해야 하는 빅 데이터 분야에서 프로비저닝은 매우 중요하다. 프로비저닝을 구축하기 위해서는 프로비전이 서버에 다양한 데몬이 필요하다. 다음은 프로비저닝을 구축하기 위해서 설치할 수 있는 서버 컴포넌트이다.



6. Hadoop Deployment Layout

시스템 구축시 Hadoop을 설치하는 경우 하드 디스크 파티셔닝에 따라서 다음과 같이 값을 설정할 수 있다. 조직에 따라 다르게 설정할 수 있으나 일반적으로 최소한 다음의 규칙에 따라서 작성하는 것이 좋다.

항목	의미	예상 경로
Hadoop 홈 디렉터리	• Apache Hadoop을 설치할 디렉터리 • 읽기 전용	/usr/local/hadoop
Datanode 데이터 디렉터리	• 실제 HDFS의 블록을 저장하는 디렉터리 • 물리 디스크에 각각의 디렉터리를 할당 • Task Tracker MapReduce 로컬 디렉터리와 같이 구성	/data1/dfs/data … /data6/dfs/data
Namenode 디렉터리	• 파일 시스템의 메타 데이터를 저장하는 디렉터리 • 서로 다른 디스크에 복제하여 기록 • 모든 디스크는 같은 용량을 갖도록 구성	/data/dfs/name
Secondary Namenode 디렉터리	• Secondary Namenode의 체크 포인트 정보를 저장하는 디렉터리	/data/dfs/secondary

MapReduce 로컬 디렉터리	• Task Tracker가 MapReduce Job 동작 시 사용하는 임시 파일을 저장하는 디렉터리 • 물리 디스크가 많을수록 성능이 향상됨 • Datanode의 데이터 디렉터리와 같이 설정	/data1/mapred/local … /data6/mapred/local
MapReduce 시스템 디렉터리	• MapReduce가 시스템 파일을 저장하기 위한 HDFS 상 경로	/hadoop/mapred/system
Hadoop 로그 디렉터리	• Hadoop 관련 데몬의 로그 파일을 저장하는 디렉터리	/var/log/hadoop
Hadoop temp 디렉터리	• Hadoop 동작 시 필요한 작고 생명주기가 짧은 파일을 보관하는 디렉터리 • 기본 : /tmp/hadoop-${user.name}	/tmp/hadoop-${user.name}
Pig 홈 디렉터리	• Apache Pig를 설치할 디렉터리 • 읽기 전용	/usr/local/pig
Hive 홈 디렉터리	• Apache Hive를 설치할 디렉터리 • 읽기 전용	/usr/local/hive
Sqoop 홈 디렉터리	• Sqoop를 설치할 디렉터리 • 읽기 전용	/usr/local/sqoop

제7절 기능 요구사항

빅 데이터는 일반적으로 애플리케이션을 개발하는 프로젝트가 아니므로 기능 요구사항을 통해 애플리케이션의 기능을 도출하는데 한계가 있으므로 일반적으로 시스템의 기능 요구사항을 포함하여 작성해야 한다.

식별자	내용	중요도	구현성	합계
FR1	시스템 및 Hadoop 리소스 모니터링	2	3	5
FR2	노드 프로비저닝	1	2	3
FR3	로그 손실 방지를 위한 이중화	5	1	6
FR4	대용량 로그의 고성능 처리를 위한 압축	3	1	4
FR5	대용량 로그 처리를 위한 Adhoc Query	5	1	6
FR6	대용량 로그 처리를 위한 프로그래밍 언어	5	1	6
FR7	대용량 로그 처리를 위한 Script Language	3	1	4
FR8	대용량 로그 처리를 위한 Scale Out 아키텍처	5	1	1
FR9	대용량 분석 결과를 데이터베이스에 저장	3	2	5
FR10	로그 데이터 수집 및 저장	3	3	6
FR11	재사용 가능한 MapReduce ETL	3	4	7
FR12	분석 기능을 제공하는 MapReduce (분석 업무에 따라 다름)	5	5	10
FR13	데이터 검증을 위한 MapReduce (분석 업무에 따라 다름)	5	5	10

대부분의 고객은 빅 데이터 프로젝트를 진행함에 있어서 왜 Apache Hadoop을 써야하는지, 그리고 왜 시스템을 과거에 다른 형태의 분산 시스템으로 구성하는지에 대한 타당성을 확인하고 싶어 하므로 이를 기능 요구사항에 포함시키는 것은 매우 중요할 수 있다.

제8절 시스템 제약사항

빅 데이터 기반 기술을 활용한 시스템을 구현할 때 전체 개발 조직과 관리 조직은 시스템의 제약사항을 숙지해야 한다. 숙지하지 않는 경우 추후 개발 생산성과 일정 지연 등 다양한

형태의 이슈가 발생한다.

식별자	내용	출처
C1	노드 장애시 배치 작업 전체에 영향을 주지 않아야 함	시스템
C2	윈도 환경에서 원활한 개발 환경 제공할 수 없음	개발 환경
C3	분석 코드가 참조하는 외부 라이브러리를 같이 패키징 해야 함	개발 환경
C4	Map과 Reduce는 단일 로그 처리시 600초 이내 수행해야 함	개발 환경
C5	로그 파일 압축시 압축하지 않은 파일과 함께 처리하지 않아야 함	시스템
C6	System.out 사용시 대용량 로그의 경우 사용을 최소화해야 함	개발 환경
C7	개발 단계에서 MRUnit을 이용하여 단위 테스트를 수행해야 함	테스트
C8	분석 결과의 정확성 확보를 위한 검증 전략을 수립해야 함	테스트
C9	단위 테스트 수행시 Code Coverage를 적용하도록 함	테스트
C10	단위 분석 작업의 임계 성능에 따라서 Pig, Hive, MapReduce를 결정할 수 있는 기준이 있어야 함	개발 환경
C11	MapReduce 워크로드에 패턴을 적용해야 함	개발환경 테스트
C12	데이터 증가에 따른 시스템 성능을 예측해야 함	테스트
C13	MapReduce 구현 패턴을 적용해야 함	개발환경

제9절 품질 속성

빅 데이터 기반 기술을 활용하는 경우 아키텍처 설계의 핵심이 되는 품질 속성(양이나 질로 관찰하여 수치로 측정할 수 있는 시스템의 특성)을 식별하고 그것을 우선순위화 하여 아키텍처 설계를 하도록 한다.

식별자	내용	관련항목	중요도	구현성	합계
QA1	데이터 이중화	FR3	5	1	6
QA2	분석 코드 유지보수성	FR11, FR12, C13	5	5	10
QA3	노드의 장애 복구	C1	3	2	5
QA4	분석 코드의 테스트 용이성	FR13, C7, C8, C9	5	5	10
QA5	개발의 용이성	FR11, C2, C7, C8, C9	4	5	9
QA6	데이터 신뢰성	FR13, C7, C8, C9	5	5	10
QA7	소스코드 재사용성	FR11, C13	3	4	7
QA8	분석 코드의 성능	FR1, FR4, FR8, FR12 C11, C12	4	4	8
QA9	노드 관리 용이성	FR2	3	2	5
QA10	분석 방법의 유연성	FR5, FR6, FR8, C10, C13	5	4	9
QA11	분석 모듈이 패키징	C3, C10	2	2	4

제10절 설계 전략

1. 처리 방법에 따른 도구 선택

대용량 데이터를 분석하고자 하는 경우 전처리, 분석, 후처리, 마이닝, 통계 등등 다양한 작업을 수행하게 된다. 데이터를 처리/분석함에 있어서 효과적으로/효율적으로 수행하기 위해서는 처리 방법에 따라서 적절하게 도구를 선정할 필요가 있다. 각 도구들은 제공하는 각각의 기능이 대체하지 않으며 상호 보완한다.

도구	권장 사용	난이도
Apache Hadoop MapReduce	• 전처리 및 후처리 작업시 사용 • 알고리즘 개발시 사용 • 최적의 성능을 구현하고자 하는 경우 • 다양한 조건에 따라 파일을 한 번에 분류해야 하는 작업에 적용 • 복잡한 수식이 표현되어 있는 경우 적용	높음
Apache Hive	• 통계 작업시 사용 • 최적의 성능을 구현하고자 하는 경우 • 필요시 User Defined Function 개발하여 확장 • Built-In Function이 지원하는 범위에서 데이터 처리시 가장 먼저 선택	매우 낮음
Apache Pig	• 전처리 및 후처리 작업시 사용 • 파일 기반 작업시 사용 • ETL 작업시 사용 • 멀티 컬럼 Join, Sort 작업시 사용 • 필요시 User Defined Function 개발하여 확장 • Hive, MapReduce에 비하여 상대적으로 처리 성능이 떨어짐	보통
Apache Mahout	• 구현된 알고리즘 적용시 사용 • 알고리즘 평가시 사용	매우 높음

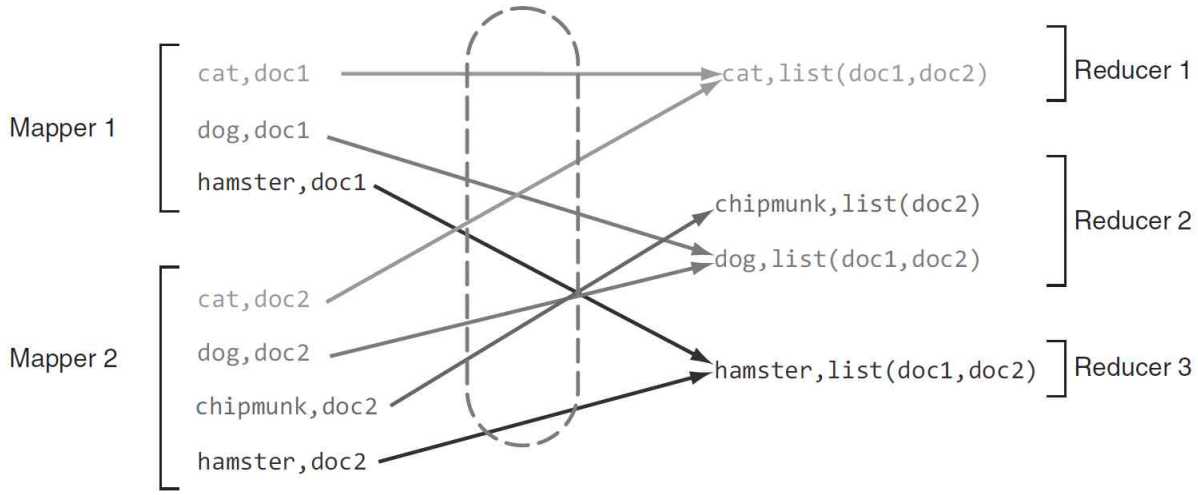
	• 알고리즘 등에서 사용하는 데이터 모델 적용시 사용 • MapReduce로 구현되어 있는 모듈이기는 하나 대용량 데이터를 마이닝 하는 경우 성능 향상이 발생할 수 있음	

다음은 이러한 다양한 구현 방법에 따라서 구현한 TStore 앱 추천 워크플로우이다. 각 단계별로 적절하게 상기 도표에 나와 있는 도구를 적절히 사용하여 데이터를 분석할 수 있다. 보통 이 과정을 제대로 수행하려면 설계 단계에서 데이터의 흐름을 정확하게 이해해야 하며 데이터 흐름을 성능, 유지보수, 테스트 용이성 등을 고려하여 분할 정복 전략을 적용해야 한다.

〈TStore의 곡 추천 분석 프로세스〉

2. 데이터 처리 흐름 설계

로그 파일을 이용하여 어떤 결과를 얻기 위해서는 데이터의 처리 흐름을 설계하는 것은 매우 중요하다. MapReduce는 Map과 Reduce가 네트워크를 중심으로 분산 처리를 하는 구조이므로 일반적은 분석 기법을 구현하는 것과는 매우 큰 차이가 있다. 따라서 MapReduce를 이용하여 데이터를 분석하거나 처리하는 경우 MapReduce의 처리 사상을 적용한 설계가 필요하고 그 핵심이 바로 데이터 흐름을 도식화 하는 것이다.



〈출처: Manning Hadoop In Practice〉

상품간 연관성을 찾아내는 알고리즘인 Association Rule Mining Algorithm을 이용하여 아이템을 추천하는 업무를 구현하는 경우 기존 시스템에서 구현이 어려운 이유는 다음과 같다.

○ 조합의 개수가 늘어날수록 급격하게 성능 저하가 발생
○ 아이템의 개수와 사용자의 구매 이력이 늘어날수록 급격하게 성능 저하가 발생

Association Rule Mining Algorithm은 데이터의 크기가 커질수록 아이템의 개수가 많아질수록 급격하게 느려진다. 느려지는 핵심 이유는 아이템의 조합을 계산하기 때문이다. 일반적으로 아이템의 조합을 계산하는 것은 아주 오랜 시간이 소요되기 때문에 기존의 처리 방식으로는 결과를 생성하기 어려우며 설령 생성한다 하더라도 매우 고가의 장비로 구현해야 한다. 따라서 이러한 알고리즘은 데이터의 크기가 증가하더라도 확장성과 성능을 보장받을 수 있는 분산 처리 방식의 빅 데이터 기반 기술을 활용하는 것이 좋다. Association Rule Mining Algorithm은 조합을 계산해야 하며 nCr로 표현하는 조합은 서로 다른 n개에서 중복 없이 r개를 택하는 경우의 수를 의미하고 다음과 같이 수식으로 표현할 수 있다. 따라서 아이템의 개수인 n이 늘어나는 경우, 조합의 개수인 r이 증가하는 경우 조합의 개수는 기하급수적으로 늘어나게 된다.

$$nCr = \frac{n!}{r!(n-r)!}$$

알고리즘에서 원하는 결과값을 얻기 위해서는 처리시 필요한 정보를 먼저 파악해야 한다. Association Rule Mining Algorithm의 입출력 정보는 다음과 같다.

○ 트랜잭션(거래수) = T

○ 지지도(Support) = P(A∩B) = N(A∩B) / N(T)

□ 많이 구매할수록 지지도는 상승

□ A와 B가 동시에 구매할 빈도수

○ 신뢰도(Confidence) = P(B|A) = P(A∩B) / P(A)

□ A를 포함하는 장바구니 중에서 B를 포함하는 빈도수

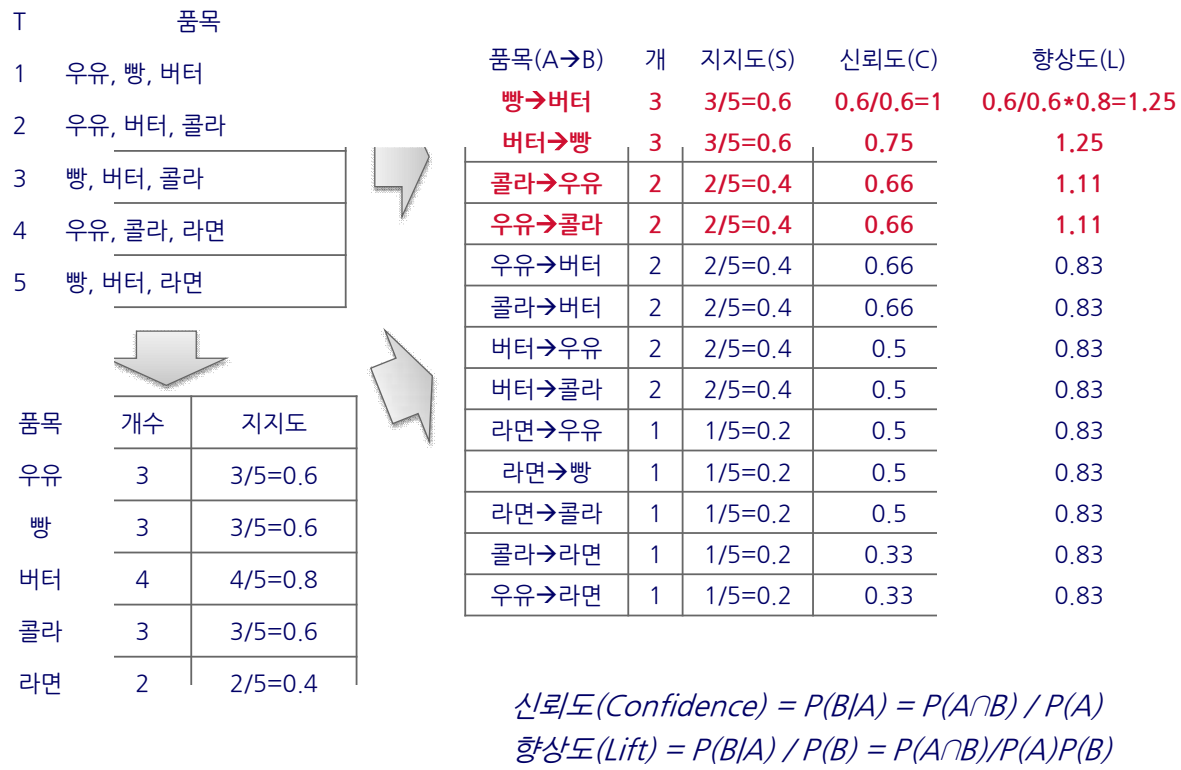
○ 향상도(Lift) = P(B|A) / P(B) = P(A∩B) / P(A)P(B)

□ A를 구매한 경우 그 트랜잭션이 B를 포함하는 경우와 B가 임의로 구매하는 경우의 비율

□ Lift 〉 1 : 높을수록 연관이 깊다

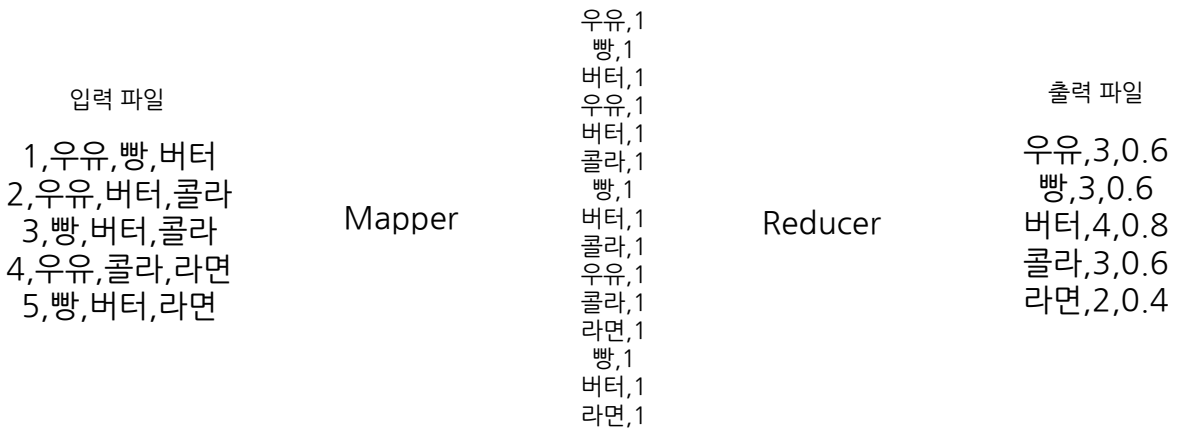
□ S,C는 얼마나 규칙이 유용한가 여부, L은 매출 향상의 기여도

추천을 하기 위해서 얻어야 하는 결과값은 support, confidence, lift이다. 전체적으로 수식을 보면 support, confidence, lift를 구하기 위해서는 어떤 상품 X에 대해서 나타날 확률을 P(X)라고 했을 때 이 P(X)는 아이템 X에 대한 건수를 트랜잭션의 수로 나눈 값이 되며, 상품 X와 Y를 동시에 구매할 확률을 P(X∩Y)라고 했을 때 이 P(X∩Y)는 아이템 X와 Y를 같이 구매한 건수를 트랜잭션의 수로 나는 값이 된다. 따라서 이 알고리즘을 적용하기 위해서는 가장 먼저 개별 아이템의 support를 계산하고 두 개의 상품을 동시에 구매하는 경우에 대해서 support, confidence, lift를 계산해야 한다.



〈Association Rule Mining Algorithm의 처리 흐름〉

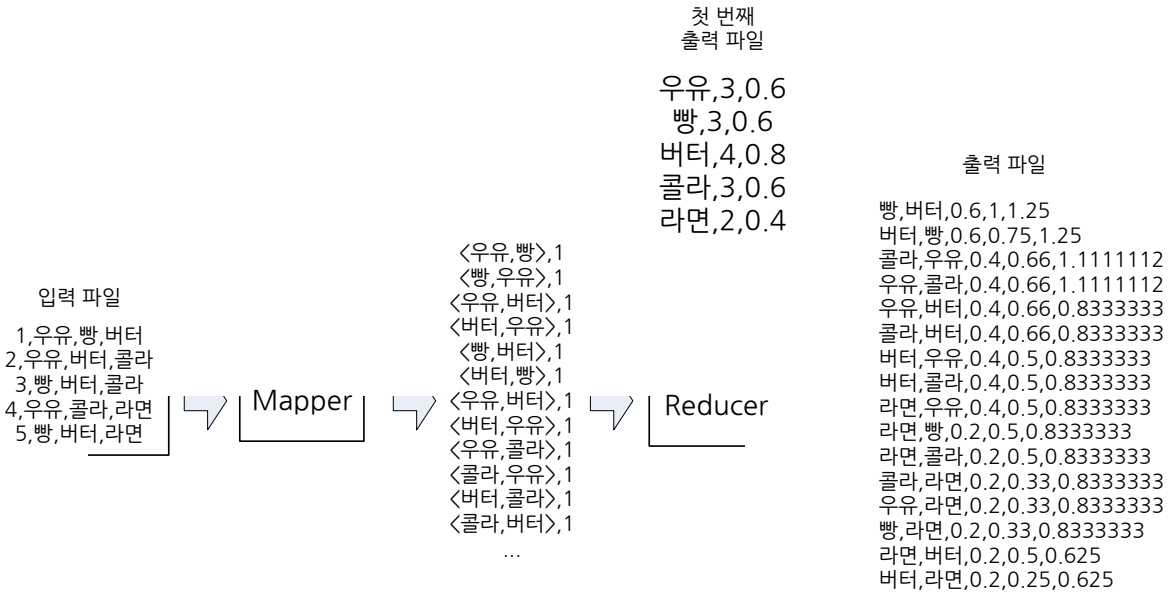
이 알고리즘을 MapReduce로 구현하기 위해서는 1단계 과정에서 각각의 아이템에 대한 나타날 확률을 계산하는 것이다. MapReduce로 다음과 같이 구현할 수 있다.



〈개별 상품에 대한 지지도를 계산하는 MapReduce 플로우〉

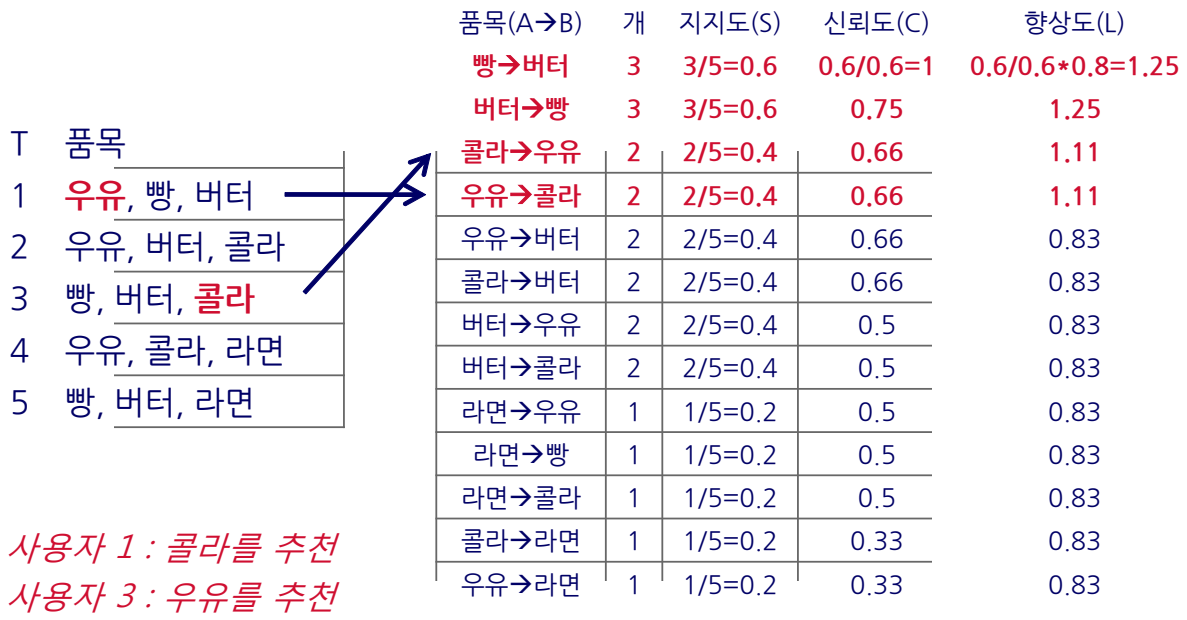
이제 support, confidence, lift를 계산하기 위해서는 2단계 과정에서 nC2에 해당하는 연산

을 수행해야 한다. 따라서 다음과 같이 MapReduce로 구현할 수 있다. 단, 여기에서 MapReduce의 제약사항으로 인하여 알고리즘을 구현하는데 큰 어려움이 발생할 수 있다. Reducer에서 첫 번째 출력 파일의 내용을 lookup하는데 아이템이 늘어나면 늘어날수록 lookup을 하기 위해서 필요한 정보는 급격하게 늘어나므로 처리하는데 있어서 메모리 문제가 발생한다. 이런 처리는 in-memory 방식을 적용해야 하나 Hadoop이 in-memory를 지원하지 않으므로 처리함에 있어서 높은 수준의 엔지니어링이 필요하다. 만약 엔지니어링을 통해서 구현할 수 없다면 알고리즘을 재설계 하거나 새로운 형식의 알고리즘을 개발해야 한다.



〈두 상품을 같이 구매하는 경우에 대한 지지도를 계산하는 MapReduce 플로우〉

알고리즘의 모든 작업이 완료되면 다음과 같이 알고리즘의 결과와 실제 로그를 결합하여 추천 목록을 생성한다. 이때 lift(향상도)는 반드시 1보다 큰 것을 사용해야 한다.



품목(A→B)	개	지지도(S)	신뢰도(C)	향상도(L)
빵→버터	3	3/5=0.6	0.6/0.6=1	0.6/0.6*0.8=1.25
버터→빵	3	3/5=0.6	0.75	1.25
콜라→우유	2	2/5=0.4	0.66	1.11
우유→콜라	2	2/5=0.4	0.66	1.11
우유→버터	2	2/5=0.4	0.66	0.83
콜라→버터	2	2/5=0.4	0.66	0.83
버터→우유	2	2/5=0.4	0.5	0.83
버터→콜라	2	2/5=0.4	0.5	0.83
라면→우유	1	1/5=0.2	0.5	0.83
라면→빵	1	1/5=0.2	0.5	0.83
라면→콜라	1	1/5=0.2	0.5	0.83
콜라→라면	1	1/5=0.2	0.33	0.83
우유→라면	1	1/5=0.2	0.33	0.83

〈Association Rule Mining Algorithm의 결과를 이용하여 추천하는 원리〉

다음은 Association Rule Mining Algorithm을 80 코어를 가진 데이터베이스에서 처리한 것과, 저비용의 x86 장비에서 분산 처리한 것의 처리 결과를 비교한 표이다.

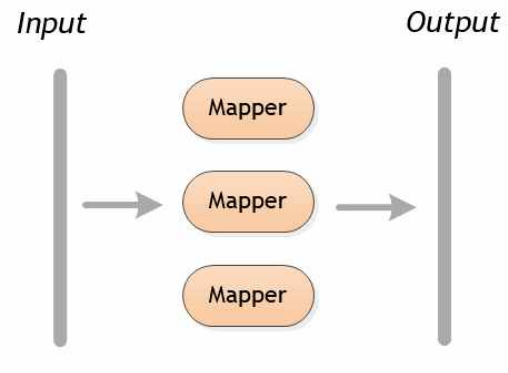
구분	Oracle 기반 머신	Hadoop 기반 머신
CPU	100%	70%
Core	80 Core	Intel 8 Core * 20 = 160 Core
처리 시간	1시간	34분
기간	1개월	1개월
상품수	120,000,000	
사용자수(T)	1,300,000	
장비 비용	6억 이상 고가 High End Server	300만원 * 20 = 6,000만원
라이선스 비용	예) Core 당 1,000만원 * 80 = 8,000만원	0
스토리지	2억 이상	

3. MapReduce 설계 패턴

MapReduce는 일반 코드와 동작 특성이 매우 상이하여 입력 파일을 처리하여 결과 파일을 생성할 때 일반적인 설계 방식으로 설계를 할 수도 없다. MapReduce를 제대로 설계하기 위해서는 우선적으로 동작 패턴을 분류해야 한다. 동작 패턴은 유지보수, 성능 등등에 따라서 서로 다른 패턴을 적용할 수 있다. 설계 시점에서 설계 패턴을 잘못 적용하는 경우 성능 저하 및 유지보수 등에 있어서 심각한 문제를 유발할 수 있으므로 설계 단계에서 각 분석 모듈별 동작 패턴을 충분히 검증하도록 한다.

2.1 Mapper

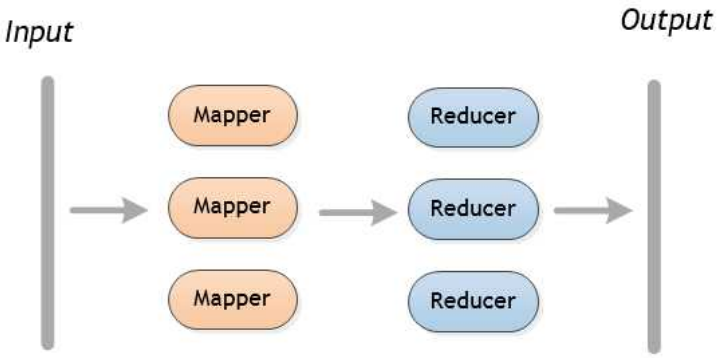
이 동작 패턴은 Mapper 하나만 사용하는 동작 패턴으로 ETL(Extract, Transform, Load) 작업을 할 때 유용하다. 특히 하나의 로그를 날짜별로 분류를 하거나 기준키로 분류하는 경우 사용할 수 있다. 고성능으로 동작하는 장점 때문에 재사용 가능한 모듈을 구현하기 용이하다.



2.2 Mapper → Reduce

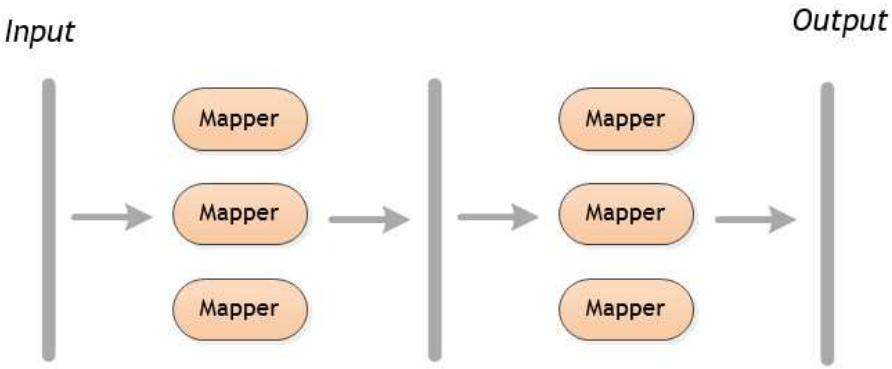
이 동작 패턴은 데이터를 취합해야 하는 경우 사용하는 패턴으로 Reducer에 취합되는 결과 데이터의 크기에 따라서 시스템의 용량이 결정될 수 있다. 예를 들어 사용자별 평균 통화 건수를 계산하는 경우 Mapper는 User Id로 출력하고, Reducer는 User Id에 대해서 통화 이력이 모이게 되는데 이때 통화건수가 많아지게 되면 Reducer는 많은 메모리가 필요하게 된다. MapReduce가 고성능이라 할지라도 Reducer는 많은 양의 데이터를 네트워크를 통해 전달하

고 Disk I/O를 발생시키므로 설계시 되도록 Reducer를 사용하지 않는 설계를 먼저 고려해야 한다.



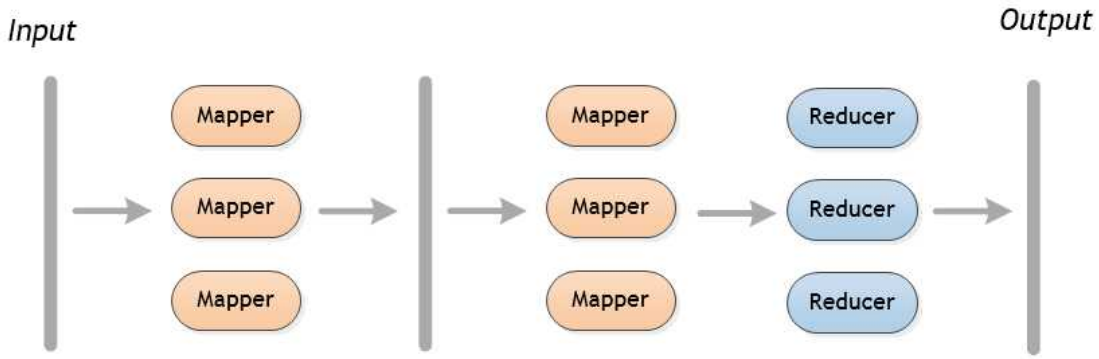
2.3 Mapper → Mapper

이 동작 패턴은 파일을 여러 단계를 거쳐서 가공하고자 할 때 사용하는 패턴이다. 예를 들면 컬럼을 삭제하고, 특정 컬럼만 필터링 하는 등의 작업을 순차적으로 적용할 때 사용한다. 또한 컬럼에 시퀀스를 생성하여 추가할 때에는 하나의 작업을 완료하기 위해서 두 번의 Mapper(Mapper별 할당된 파일의 ROW 수를 측정하고 파일에 시퀀스를 부여)를 사용하여 처리해야 한다. 보통 하나 이상의 ETL 작업을 연계하는 경우 이러한 구현 패턴을 적용할 수 있다.



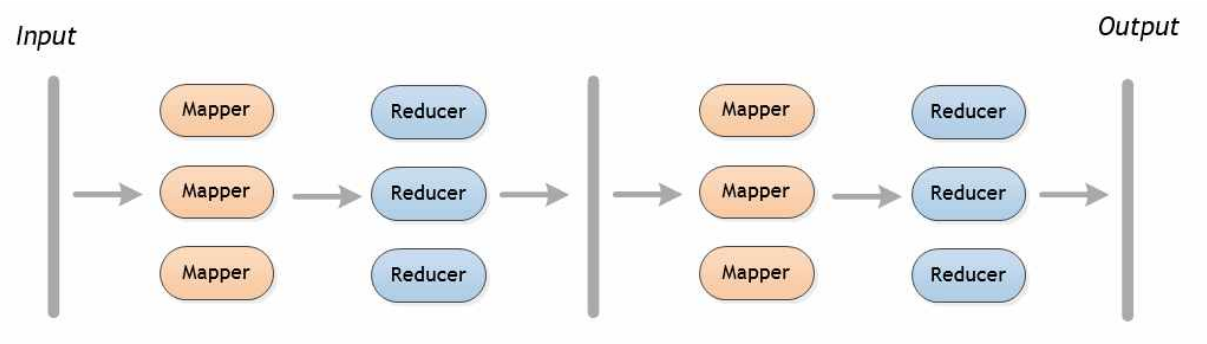
2.4 Mapper → Mapper → Reduce

이 동작 패턴은 데이터를 분류하고 그 데이터를 취합하는 등의 작업을 수행할 때 사용한다.



2.5 Mapper → Reduce → Mapper → Reduce

이 동작 패턴은 Unsupervised Algorithm 등에서 많이 활용하는 방법으로 임계치에 도달할 때 까지 동일한 작업을 반복하는 경우(예; K-Means) 보통 적용한다. Collaborative Filtering 이나 FP-Tree 등의 알고리즘은 그 결과를 생성하기 위해서 지속적으로 Group By 작업을 수행하면서 데이터를 취합하는 과정을 해야 한다. 가장 성능이 느린 작업이므로 성능 확보를 위해서 기존 알고리즘의 구현방식을 새로이 개선하거나 새로운 알고리즘을 개발해야 한다.



4. Hive & Pig의 User Defined Function (UDF) 적용 전략

동일한 결과를 얻기 위해서 MapReduce 구현 패턴을 적용할 수 있지만 빠르게 프로토타이핑을 하고 고성능, 유연성을 확보하기 위해서 UDF를 사용할 수 있다. Hive, Pig에서 제공하지만 MapReduce에서는 제공하지 않는 경우 다음과 같이 원하는 기능을 추가하여 실제로 사용가능하다.

```
package org.openflamingo.hive;

import org.apache.hadoop.hive.ql.exec.Description;
import org.apache.hadoop.hive.ql.exec.UDAF;
import org.apache.hadoop.hive.ql.exec.UDAFEvaluator;

import java.util.ArrayList;

/**
 * Group First Item Hive UDAF.
 * 이 UDAF는 Aggregation한 결과 데이터에서 가장 첫번째 항목을 선택하는 기능으로 중복된 데이터가
 * 존재하더라도 모두
 * List에 담아서 처리하게 되므로 JVM Heap 소비가 발생한다. 따라서 Aggregation의 단위 건수가 많고 해당
 * 구분키가 중복된 경우 이 메소드를 사용하기 보다는 중복 제거 기능을 제공하는 Set을 사용할 것을 권장한다.
 * <p/>
 * <ul>
 *   <li>Mapper : iterate - terminatePartial</li>
 *   <li>Reducer : init - merge - terminate</li>
 * </ul>
 *
 * @author Edward KIM
 * @since 0.1
 */
@Description(
        name = "groupFirst",
        value = "_FUNC_(arr) - Return a first item in aggregation. arr can be a array.",
        extended = "Example:\n"
                + "  > SELECT _FUNC_(groupFirst(contractId)) FROM Log GROUP BY contractId;\n"
                + "  '192837182'"
)
public class UDAFFirst extends UDAF {

    /**
     * 기본 생성자.
     */
    public UDAFFirst() {
    }

    /**
     * Aggregation을 수행하는 실제 클래스.
     * Hive는 UDAFEvaluator를 구현하는 UDAF의 모든 클래스를 자동을 찾는다.
     */
    public static class UDAFGroupFirstEvaluator implements UDAFEvaluator {

        ArrayList<String> data;
```

```
public UDAFGroupFirstEvaluator() {
    data = new ArrayList<String>();
}

/**
 * Aggregation의 상태값을 초기화한다.
 */
public void init() {
    data.clear();
}

/**
 * 원본 데이터의 1개의 ROW를 반복한다.
 * 이 UDAF는 문자열로 된 인자를 N개 수용하며 단일 문자열을 처리하고자 하면
 * 단일 문자열 인자를 사용하도록 한다.
 *
 * 이 메소드는 항상 <tt>true</tt>를 반환한다.
 */
public boolean iterate(String[] elements) {
    if (elements != null) {
        StringBuilder builder = new StringBuilder();
        for (String element : elements) {
            builder.append(element);
        }
        data.add(builder.toString());
    }
    return true;
}

/**
 * 부분 Aggregation을 마무리하고 상태를 반환한다.
 */
public ArrayList<String> terminatePartial() {
    return data;
}

/**
 * 부분 Aggregation을 병합한다.
 * 이 함수는 항상 같은 유형의 값들을 가진 단일 인자를 가지도록 해야 하며
 * terminatePartial() 메소드의 결과값을 받는다.
 *
 * 이 메소드는 항상 <tt>true</tt>를 반환한다.
 */
public boolean merge(ArrayList<String> elements) {
    if (elements != null) {
        data.addAll(elements);
```

```
            }
            return true;
        }

        /**
         * Aggregation을 마무리하고 최종 결과를 반환한다.
         * 최종 결과를 생성할 때에는 가장 첫번째 아이템만 추출한다.
         *
         * @return Group First UDAF의 최종 결과
         */
        public String terminate() {
            return data.get(0);
        }
    }

}
```

이와 같은 경우 다음과 같이 Hive QL을 사용할 때 다음과 같이 확장할 수 있다.

```
set mapred.reduce.tasks=100;
set mapred.job.name='Monthly Statistics (${YESTERDAY})';

add jar mapreduce-template-1.0.jar
CREATE TEMPORARY FUNCTION groupFirst AS 'org.openflamingo.hive.UDAFFirst';
CREATE TEMPORARY FUNCTION groupLast  AS 'org.openflamingo.hive.UDAFLast';

SELECT groupFirst(userSequence) FROM User GROUP BY username; // 실제 통계
```

MapReduce는 framework만 존재하기 때문에 특정한 기능(예; JOIN, SORT, PIVOT)을 수행하기 위해서는 MapReduce를 기반으로 별도의 기능을 확장해야 한다. 특히 이러한 기능이 대용량의 로그를 처리해야 하는 환경에서는 완전히 새롭게 개발을 해야 하는 이슈가 발생할 수 있으므로 Hive나 Pig의 구현 메커니즘을 적용하여 구현하는 전략은 결코 나쁜 전략이 아니며 오히려 더욱더 권장해야할 구현 전략이다.

제11절 테스트 및 검증 전략

빅 데이터는 데이터를 다루기 때문에 AS-IS를 TO-BE로 전환하거나 신규 시스템이라 할지라도 데이터의 신뢰성을 보장하는 것은 굉장히 중요하다. 따라서 신뢰성을 확보하기 위한 전략을 수립할 필요가 있다.

1. 단위 테스트 및 데이터 검증 전략

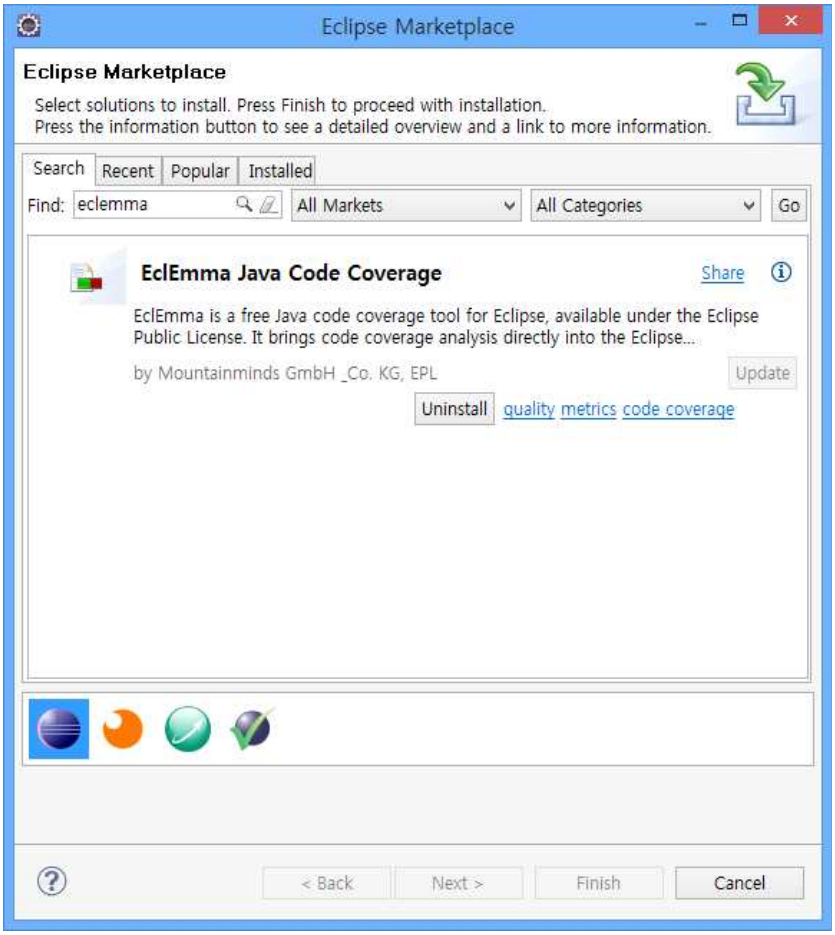
애플리케이션을 구현할 때 단위 테스트를 진행하여 품질을 확보하는 것처럼 MapReduce로 구현한 데이터 분석 모듈도 단위 테스트를 수행할 수 있다. Apache MRUnit[4]은 JUnit[5]을 기반으로 동작하는 단위 테스트 프레임워크로서 구현한 각 모듈에 대해서 처리 결과가 올바르게 출력되는지 확인할 때 사용한다. 다음의 Group By를 수행하는 MapReduce ETL의

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mrunit.mapreduce.MapReduceDriver;
import org.junit.Before;
import org.junit.Test;

public class GroupByMapReduceTest {

    private Mapper mapper;
    private Reducer reducer;
    private MapReduceDriver driver;

    @Before
    public void setUp() {
        mapper = new GroupByMapper();
        reducer = new GroupByReducer();
        driver = new MapReduceDriver(mapper, reducer);
    }

    @Test
    public void groupBy() {
```

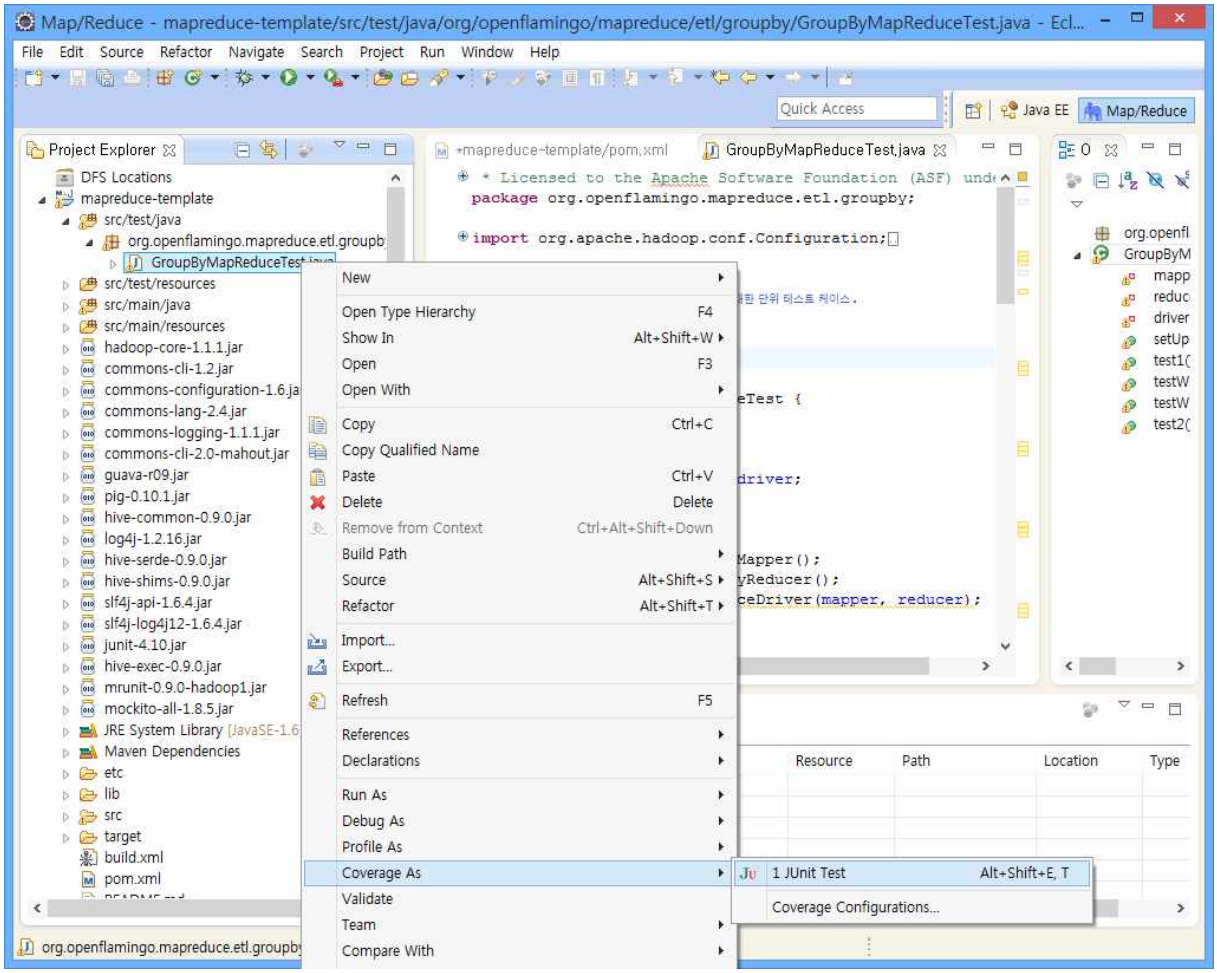
4) MRUnit Official Site : http://mrunit.apache.org
5) JUnit Official Site : http://www.junit.org

```
        Configuration conf = new Configuration();
        conf.set("inputDelimiter", ",");
        conf.set("keyValueDelimiter", ",");
        conf.set("valueDelimiter", ",");
        conf.set("allowDuplicate", "false");
        conf.set("allowSort", "false");
        conf.set("groupByKey", "0");

        driver.setConfiguration(conf);

        driver.withInput(new LongWritable(1), new Text("홍길동,a,b"));
        driver.withInput(new LongWritable(2), new Text("홍길동,b"));
        driver.withOutput(NullWritable.get(), new Text("홍길동,a,b"));
        driver.runTest();
    }
}
```

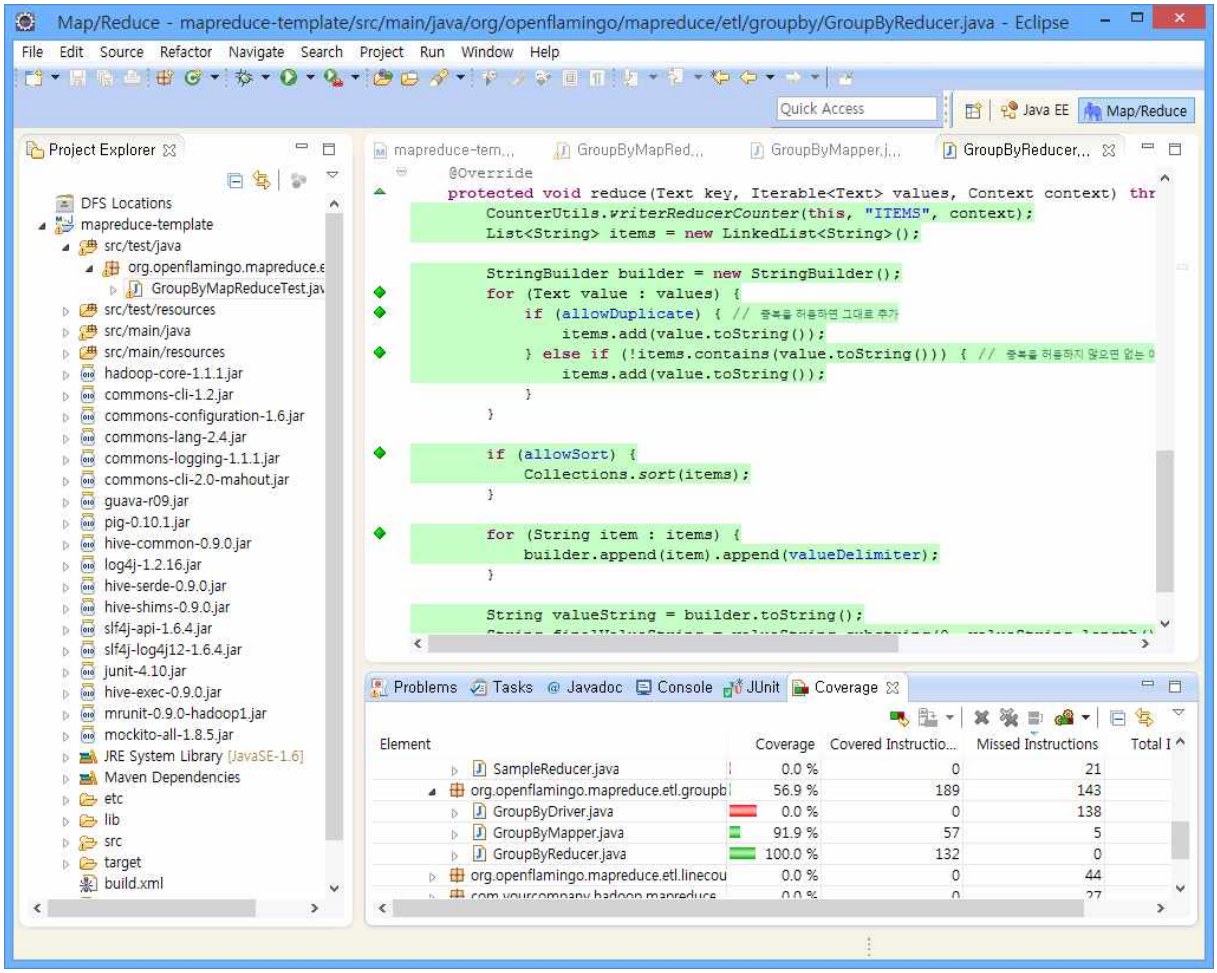
또한 분석 요구사항에 따라서 다양한 조건에 따라서 분기를 수행하고 로그의 포맷에 따라서 다양한 처리를 하는 경우 대용량 데이터를 입력으로 사용하는 통합 및 시스템 테스트는 검증하는 시간을 확보하기 매우 어렵고, 데이터 검증도 매우 어렵다. 따라서 EclEmma와 같은 Code Coverage를 도입하도록 한다. 다음은 Eclipse의 Marketplace를 통해 무료로 설치할 수 있는 EclEmma의 설치 화면이다.



이제 MRUnit으로 작성한 단위 테스트 케이스를 다음과 같이 Code Coverage를 이용하여 테스트를 수행한다.



그러면 다음과 같이 작성한 단위 테스트가 실행되고 실제 구현한 MapReduce가 어떻게 실행되었는지 확인할 수 있다. 입력 데이터에 따라서 if 조건을 채우지 못하는 경우도 발생하므로 데이터의 자료형, NULL 여부, 해당 값이 존재 유무 등을 고려하여 실제 데이터를 구성하고 테스트한다. 이를 개발단계에서 수행하지 않으면 추후 통합/시스템 테스트시 많은 문제가 발생하게 되고 데이터의 신뢰성도 보장하지 못하게 된다.



2. MapReduce 데이터 처리 패턴 분류

MapReduce는 데이터를 처리하는 과정이 다음과 같이 일정한 패턴을 갖게 된다. 이러한 패턴을 사전에 수집하여 그와 유사한 패턴을 찾아서 성능 시험을 할 수 있다. 예를 들면 전처리 MapReduce는 대량의 데이터를 읽어서 대량의 데이터를 생성하는 MapReduce이다. 이 과정을 시뮬레이션하려면 이와 유사한 처리 패턴을 찾아야 한다. 이와 유사한 처리 패턴으로 DFS IO 처리 패턴을 확인하면 이 패턴을 적용한 벤치마킹을 실제로 수행하여 전처리 과정의 성능을 예측해야 한다.

Workload	Resource	Execution Pattern					Job Type
		Input	Map	Intermediate	Reduce	Output	
TeraSort	Map : CPU Bound Reduce : I/O Bound		Map		Reduce		Large Job
WordCount	CPU Bound		Map		Reduce		Group By Job
DFS IO	I/O Bound	R	Map	W	Reduce		Read/Write Job
mrbench	Job Bound		Map		Reduce		Many Small Job
전처리 MapReduce	I/O Bound		Map		Reduce		Classification Job
분석 MapReduce	Map : I/O Bound Reduce : CPU Bound		Map		Reduce		Group By & Aggregation Job

3. 성능 테스트 전략

데이터 처리 패턴에 따라서 실제로 DFS IO를 수행할 때 데이터의 생성주기, 사용자수, 단위 로그의 크기, 배치 처리 주기에 따른 총 로그의 예상 크기 등을 고려하여 테스트 파라미터를 결정한다. 그리고 테스트 후에 전처리 MapReduce의 처리 예상 시간을 판단하여 시스템의 용량을 산정한다.

테스트 조건							테스트 파라미터		테스트 결과	
DFSIO Type	생성주기(분)	User	Running(2H)	Raw Size(Bytes)	예상 1일 크기(Bytes)	예상 1일 크기(M)	Files	Size(M)	Total Time(s)	Throughput(M)
Read	5	100,000	2	300	3,600,000,000	3,433.23	288	11.92	103	64.17
	4	100,000	2	300	2,880,000,000	2,746.58	288	9.54	100.8	59.07
	3	100,000	2	300	2,160,000,000	2,059.94	288	7.15	99.8	46.09
	2	100,000	2	300	1,440,000,000	1,373.29	288	4.77	99.8	41.27
	1	100,000	2	300	720,000,000	686.65	288	2.38	97.8	35.81
Write	5	100,000	2	300	3,600,000,000	3,433.23	288	11.92	103	12.1
	4	100,000	2	300	2,880,000,000	2,746.58	288	9.54	103.9	12.1
	3	100,000	2	300	2,160,000,000	2,059.94	288	7.15	101.9	11.1
	2	100,000	2	300	1,440,000,000	1,373.29	288	4.77	101.8	13.3
	1	100,000	2	300	720,000,000	686.65	288	2.38	98.8	11.3
Read	5	500,000	2	300	18,000,000,000	17,166.14	288	59.60	100	77.77
	5	1,000,000	2	300	36,000,000,000	34,332.28	288	119.21	111	39.91
	5	5,000,000	2	300	180,000,000,000	171,661.38	288	596.05	278	39.32
	5	10,000,000	2	300	360,000,000,000	343,322.75	288	1,192.09	461	38.64
	5	20,000,000	2	300	720,000,000,000	686,645.51	288	2,384.19	817	40.21
Write	5	500,000	2	300	18,000,000,000	17,166.14	288	59.60	174	8.75
	5	1,000,000	2	300	36,000,000,000	34,332.28	288	119.21	316	6.54
	5	5,000,000	2	300	180,000,000,000	171,661.38	288	596.05	1331	5.9
	5	10,000,000	2	300	360,000,000,000	343,322.75	288	1,192.09	2575	5.82
	5	20,000,000	2	300	720,000,000,000	686,645.51	288	2,384.19	5079	5.77

다음은 상기 도표에 표시되어있는 값을 결정하는 방법이다. 1일 배치 처리를 기준으로 한다.

○ 단위 로그 생성 건수 = 생성주기 × Running = 5분당 1개 × 2시간 = 120건

○ 예상 1일 크기 = 단위 로그 생성 건수 × User × Raw Size

= 120 × 100,000 × 300

= 3,600,000,000 Byes

○ Files = 시간당 생성 건수 × 24 (1일) = 288

○ Size = 1일 예상 크기 / Files = 11.92M

이렇게 결정된 파라미터를 기준으로 DFSIO[6] 벤치마크 테스트를 다음가 같이 진행할 수 있다.

```
#hadoop jar $HADOOP_HOME/hadoop-*test*.jar TestDFSIO -clean
#hadoop jar $HADOOP_HOME/hadoop-*test*.jar TestDFSIO -write -nrFiles 288 -fileSize 11.92
#hadoop jar $HADOOP_HOME/hadoop-*test*.jar TestDFSIO -read -nrFiles 288 -fileSize 11.92
```

이 DFSIO 테스트를 수행하고 나면 다음과 같이 수행한 시간을 알 수 있으며 이를 상기 도표에 기록한다.

6) DFSIO 소스코드 : $HADOOP_HOME/src/test/org/apache/hadoop/fs/TestDFSIO.java

```
----- TestDFSIO ----- : write
           Date & time: Fri Apr 08 2011
       Number of files: 1000
Total MBytes processed: 1000000
     Throughput mb/sec: 4.989
Average IO rate mb/sec: 5.185
 IO rate std deviation: 0.960
    Test exec time sec: 1113.53

----- TestDFSIO ----- : read
           Date & time: Fri Apr 08 2011
       Number of files: 1000
Total MBytes processed: 1000000
     Throughput mb/sec: 11.349
Average IO rate mb/sec: 22.341
 IO rate std deviation: 119.231
    Test exec time sec: 544.842
```

DFSIO 테스트 결과에 표시되어 있는 Throughtput은 다음과 같이 계산할 수 있다.

$$Throughtput(N) = \frac{\sum_{i=0}^{N} filesize_i}{\sum_{i=0}^{n} time_i}$$

DFSIO 테스트 결과에 표시되어 있는 Average IO Rate는 다음과 같이 계산할 수 있다.

$$Average\,IO\,rate(N) = \frac{\sum_{i=0}^{N} rate_i}{N} = \frac{\sum_{i=0}^{N} \frac{filesize_i}{time_i}}{N}$$

제2장 데이터 수집

제1절 로그 수집 전략

빅 데이터 기반 로그 분석 시스템을 구축할 때 첫 시작은 항상 데이터를 수집하는 일이다. 데이터를 수집하는 방법은 데이터의 생명 주기와 외부 시스템의 데이터 생성 주기에 따라서 서로 다른 전략을 구사할 수 있으므로 환경에 따라서 적절하게 전략을 선택하도록 한다.

전략	장점	단점	오픈소스
File Uploader	- 단순한 구조 - 덤프형 로그 파일 기반	- 배치 처리만 지원 - 메시지 변환 불가 - 메시지 전처리 불가	Flamingo HDFS File Uploader File Slurper
Log Aggregator	- 실시간 처리 - 메시지 변환 - 분산 처리 - 덤프 파일 실시간 처리	- 복잡한 시스템 구성 - 메시지 손실 발생 - 성능 및 안정성에 대한 다양한 정책	Apache Flume Facebook Scribe
Message Service	- 실시간 처리 - 분산 처리 - 토픽 및 큐 기반 - 메시지 손실 최소화	- 복잡한 시스템 구성 - 메시지 전처리 불가	Apache Kafka
Event Streaming	- 실시간 처리 - 메시지 변환 - 분산 처리	- 복잡한 시스템 구성 - 서로 상이한 확장성 - 장애 발생시 메시지 처리 문제 발생 가능성	Esper CEP Apache S4 Storm

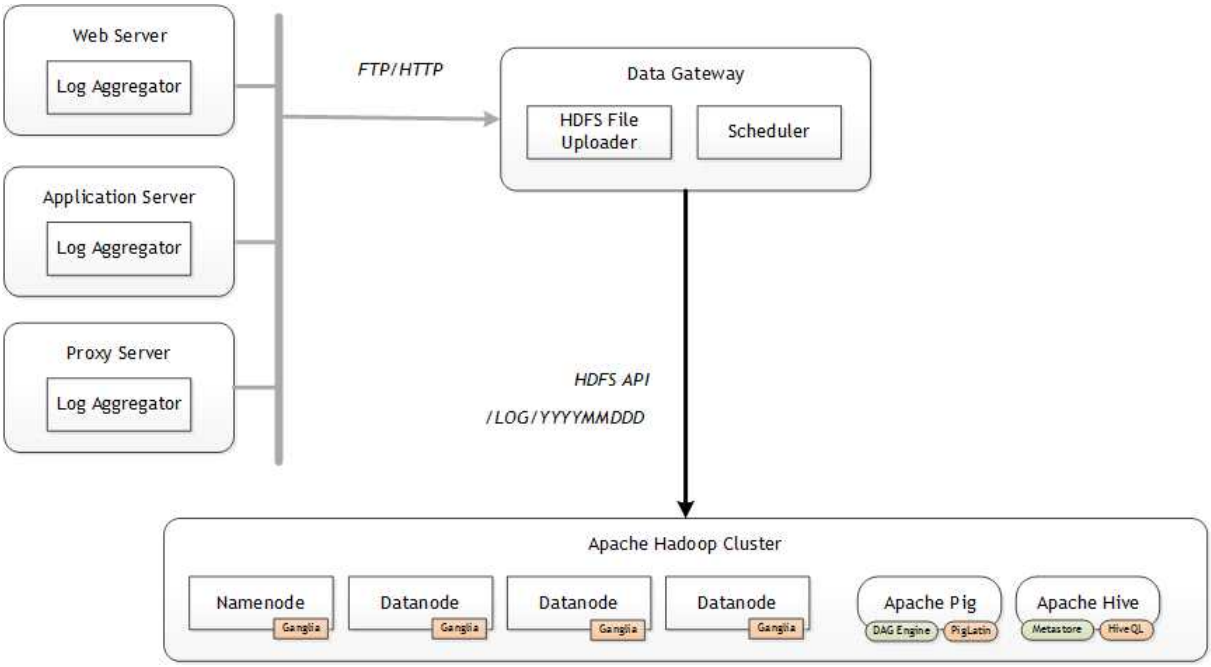
제2절 로그 수집

1. HDFS File Uploader

일반적으로 대부분의 시스템은 외부 시스템을 통해서 로그를 수집하는 경우 파일을 선호한다. 특히 FTP 서버 및 클라이언트 기반 파일 송수신은 그 목적에 따라서 현장에서 가장 많이 사용하는 방법이다. FTP는 처리하는 방법에 따라서 일반적으로 다음 두 가지 연동 방법이 있다.

- FTP Put
  - O 외부 시스템에서 Hadoop Cluster가 있는 시스템으로 업로드 한다.
  - O 연동시 외부 시스템에서 해당 시스템에 접근할 수 있도록 방화벽을 열어야 한다.
  - O 외부 시스템에 파일을 업로드할 수 있도록 FTP 계정을 생성해야 한다.
  - O 방화벽에 해당 서버의 IP 주소와 포트를 등록하여 보안 처리한다.
- FTP Get
  - O Hadoop Cluster가 있는 시스템 외부 시스템으로 파일을 다운로드한다.
  - O 연동시 외부 시스템에서 접근 가능한 계정을 생성해야 한다.
  - O 연동시 외부 시스템에서 해당 시스템에 접근할 수 있도록 방화벽을 열어야 한다.
  - O Hadoop Cluster가 있는 시스템에서 외부 시스템에 접근하기 위한 서버를 단일화 하여 보안에 대응한다.

일반적으로 현업에서 가장 선호하는 방법은 외부 시스템 입장에서 보안 정책을 변경하지 않고도 시스템 연동을 할 수 있기 때문에 FTP Put을 선호한다. FTP Put 정책은 외부 시스템에서 파일을 밀어 넣어주는 방식이므로 자동화 하는데 있어서도 편리한 장점이 있다. 반면 FTP Get은 외부 시스템과 연동시 외부 시스템의 보안 정책을 따라야 하기 때문에 연동하는데 시간이 더 소요되는 특징이 있다. 주로 Hadoop Cluster가 있는 시스템이 후에 구축된 시스템일 가능성이 매우 높기 때문에 외부 시스템에서는 FTP Put을 선호한다.

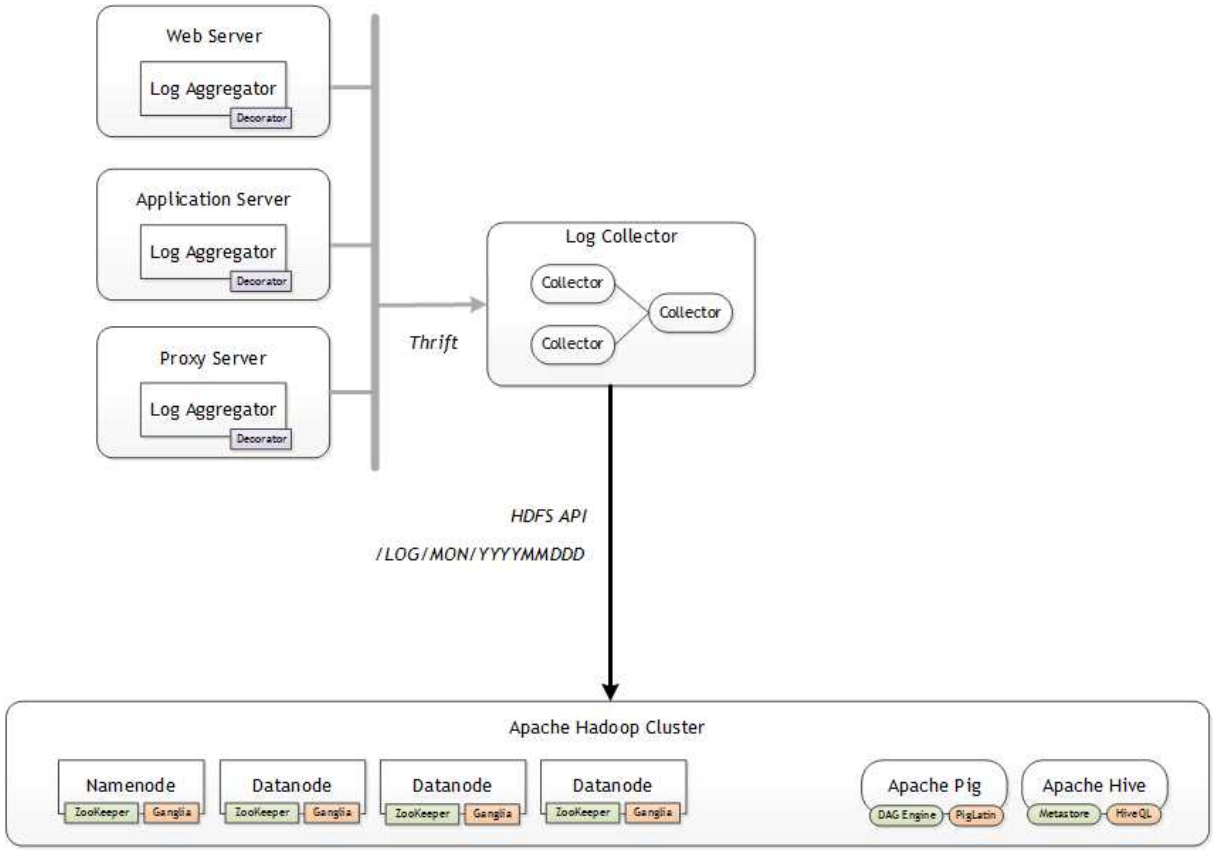


다만 이 방식은 통상적으로 L4 스위치를 통한 부하 분산 처리가 되어 있지 않은 경우 한쪽 노드에만 파일을 적재하기 때문에 장애 발생시 파일을 처리하지 못하는 문제가 발생하므로 다음과 같은 이중화 정책을 통해 대응한다.

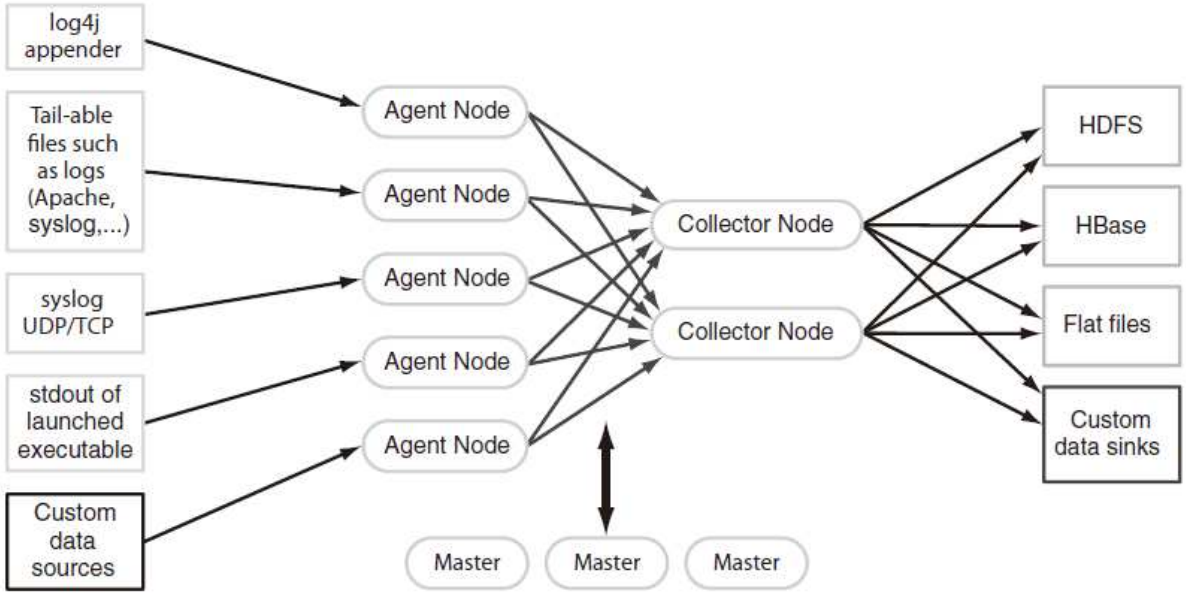
- L4 스위치로 FTP 서버를 다수 구성하여 부하를 분산시킨다.
- FTP 서버는 NAS 등을 통해 스토리지를 통합하여 스토리지 장애에 대응한다.

2. Log Aggregator

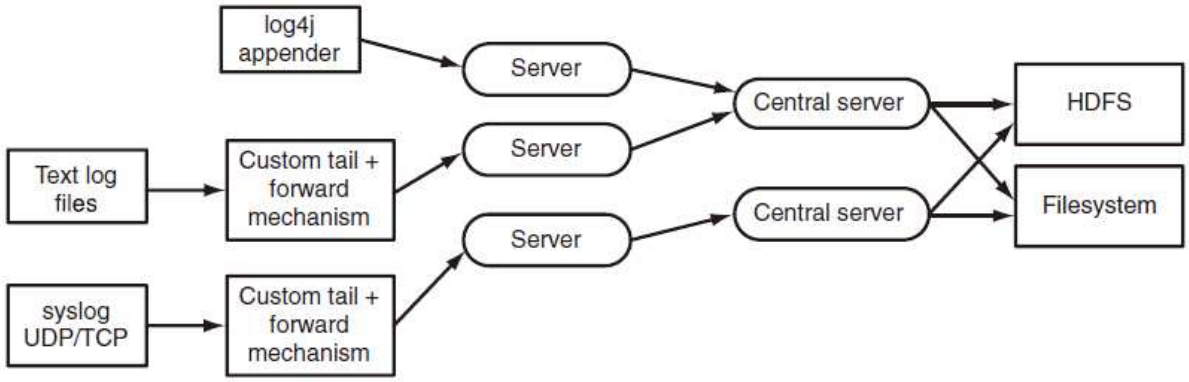
로그 수집기를 통한 로그 수집 방식은 실시간 로그 메시지 전달 및 수집, 메시지 손실 최소화, 메시지 변환 등을 수행할 수 있는 장점이 있으나 반면 Apache Flume의 경우 Apache ZooKeeper와 같은 Distributed Coordinator의 도움을 받아야 하고, Scribe의 경우 Java 기반으로 구성되어 있지 않으므로 Unix 머신에서 동작이 어렵다는 단점이 있다. 특히 Log Aggregator는 로그를 수집함에 있어서 로그 메시지를 전달하는 agent를 해당 서버에 설치해야 하므로 외부 연동 시 조직 간의 이해관계 먼저 해소해야 하는 문제가 있지만 메시지를 agent 수준에서 변환할 수 있다는 점, QoS 정책을 변경할 수 있다는 점이 장점이다.



다음은 Apache Flume의 아키텍처로써 Apache Flume은 Agent와 Collector가 Thrift 프로토콜을 사용하여 연동되며 Agent는 다양한 Source에서 로그를 수집하여 Collector로 전달하고 Collector는 수집한 로그를 HDFS나 파일로 기록한다.



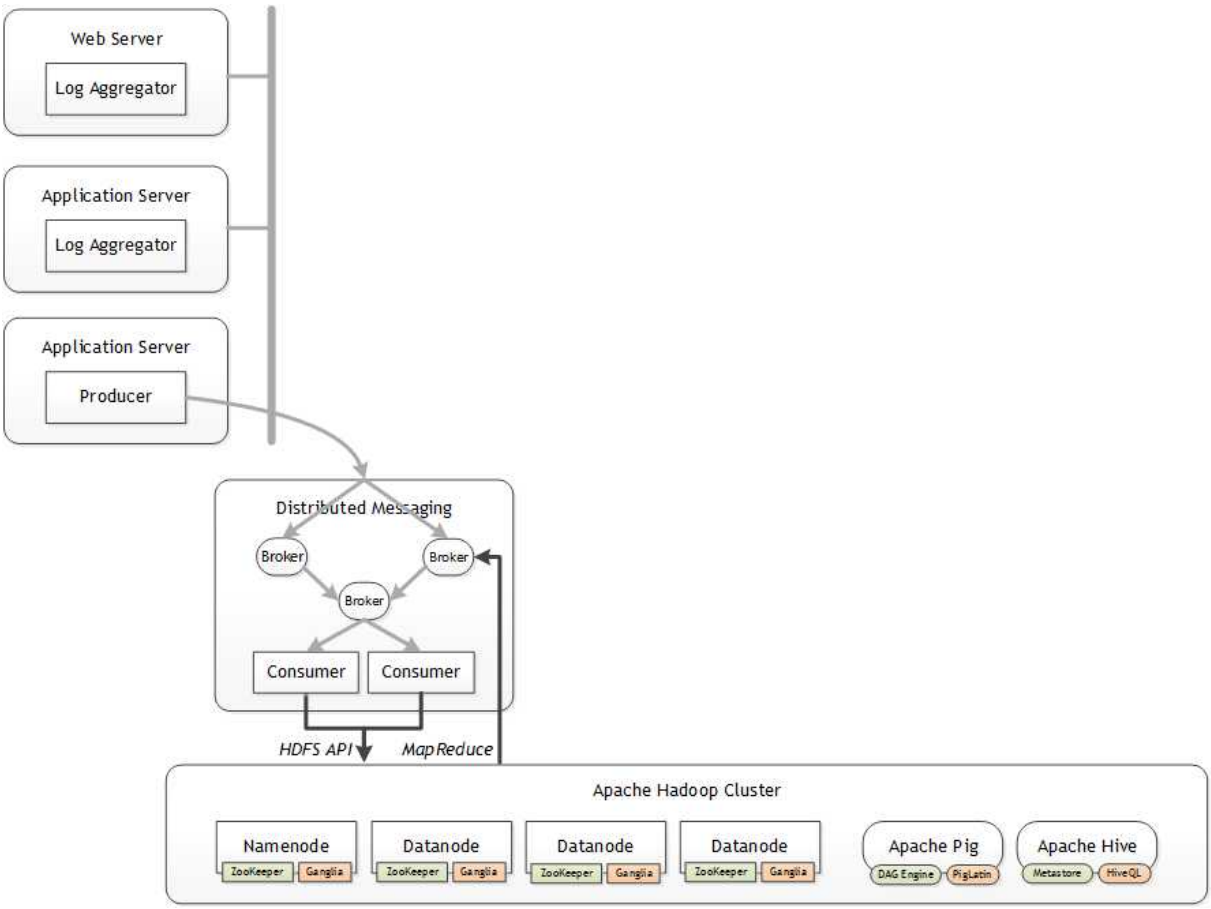
다음은 Facebook Scribe의 아키텍처로써 Scribe는 가장 단순한 아키텍처를 지향한다. 다양성은 부족하지만 Twitter와 Facebook에서 사용하고 있을 만큼 성능과 안정성은 타 오픈소스를 능가한다.



3. Message Queue

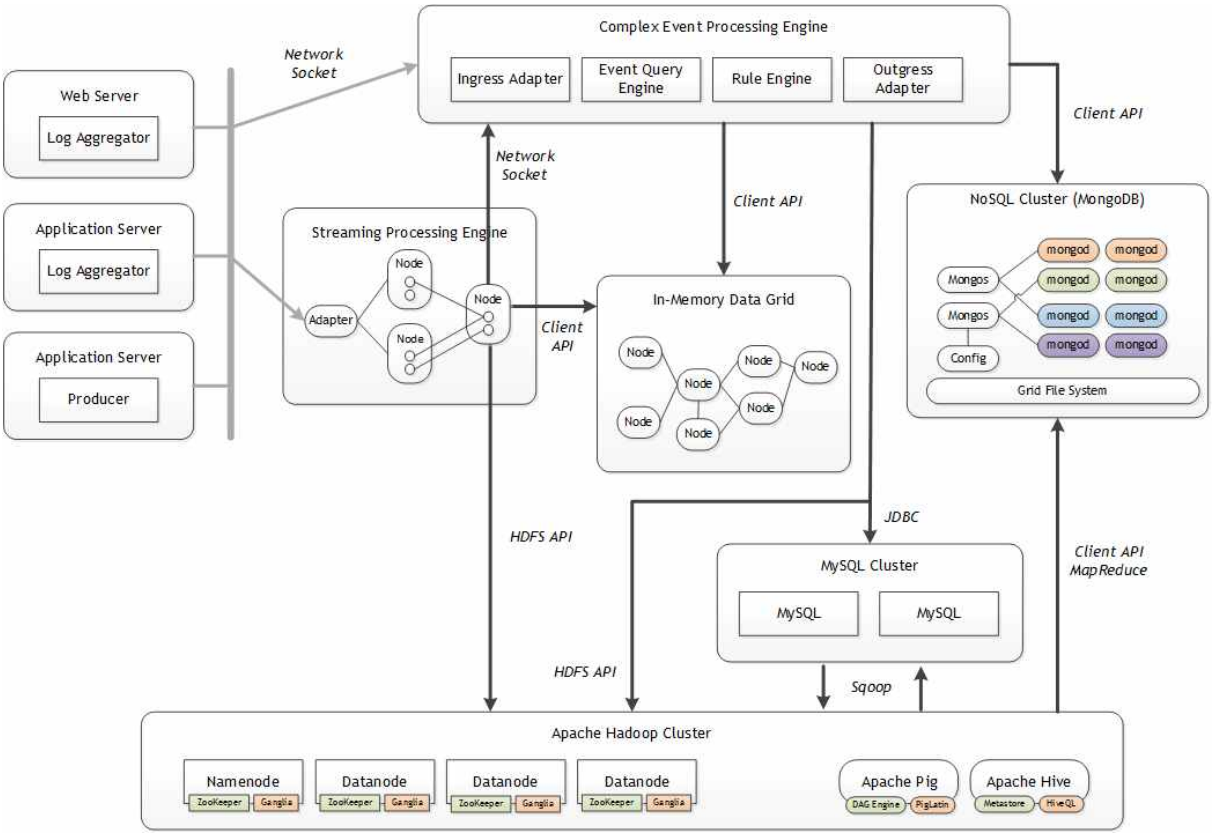
Message Queue 방식은 전통적인 메시지 송수신 시스템 기반으로서 오랫동안 검증된 시스템이라는 것과 메시지의 송수신에 최적화 되어 있다는 장점이 있지만 메시지 전처리가 불가능하고 Apache Kafka의 경우 Apache ZooKeeper와 같은 Distributed Coordinator의 도움을 받아야 한다는 단점이 있다. 특히 Apache Kafka의 경우 국내 적용 사례가 거의 없다는 점이

도입시 걸림돌이 될 수 있다. 하지만 실제로 국내 적용한 사례를 살펴보면 Apache Kafka의 경우 안정적인 메시지 송수신에 대해서 검증되었음을 확인할 수 있다.



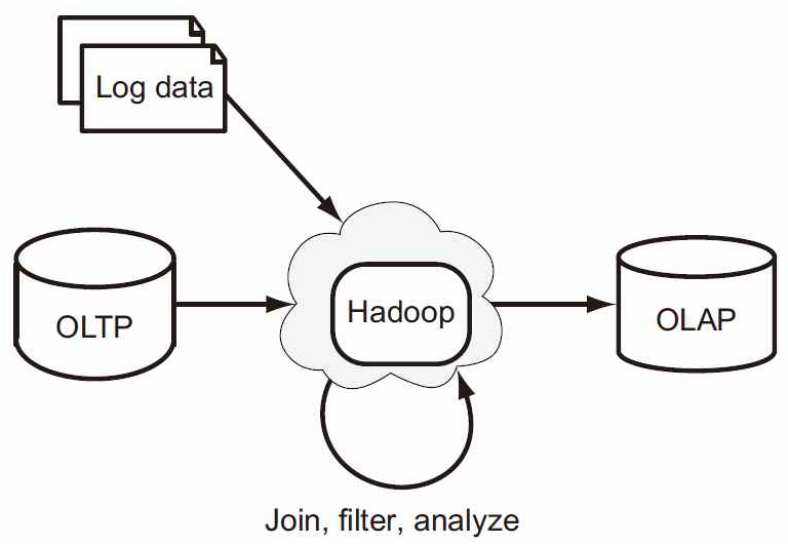
4. Event Streaming

이벤트 스트리밍은 실시간 이벤트 및 로그 수집에 있어서 가장 완전한 솔루션이기는 하나 구성이 복잡하고 많은 장비를 준비해야 하며, 동시에 각각의 오픈소스의 기능적/비기능적 요구사항이 매우 상이하여 오픈소스를 선정하는데 있어서 어려운 점이 있다. 다만 도입시 서비스의 수준을 한 차원 높일 수 있게 된다. 2013년부터는 현장에서 리얼타임 이벤트 스트리밍 요건이 매우 중요한 시기가 도래할 것이기 때문에 특별히 이러한 데이터 수집 방법에 대한 기술적 능력을 향상 시켜야 한다.

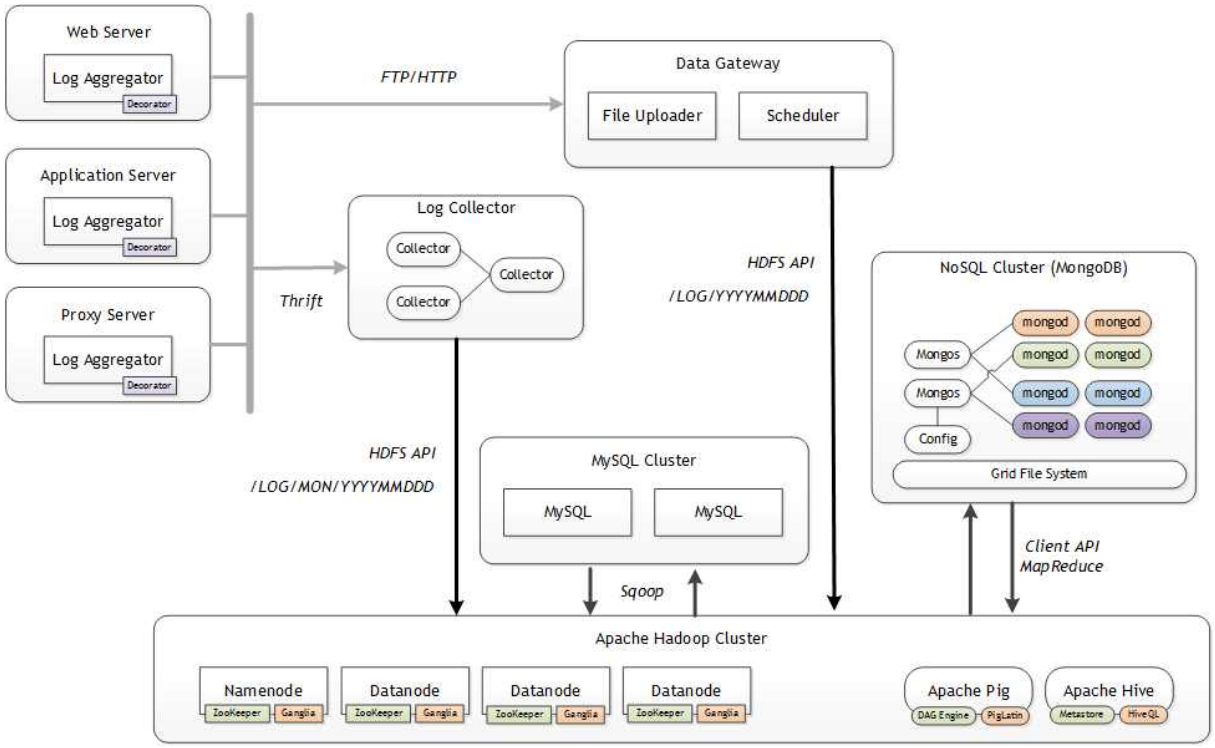

Complex Event Processing Engine
Ingress Adapter
Event Query Engine
Rule Engine
Outgress Adapter
Network Socket
Client API
Web Server
Log Aggregator
Application Server
Log Aggregator
Application Server
Producer
Streaming Processing Engine
Adapter
Node
Node
Node
Node
Client API
In-Memory Data Grid
Node
Node
Node
Node
Node
Node
NoSQL Cluster (MongoDB)
Mongos
Mongos
Config
mongod
Grid File System
HDFS API
JDBC
MySQL Cluster
MySQL
MySQL
Client API MapReduce
HDFS API
Sqoop
Apache Hadoop Cluster
Namenode
Datanode
Datanode
Datanode
ZooKeeper
Ganglia
Apache Pig
DAG Engine
PigLatin
Apache Hive
Metastore
HiveQL

제3절 Sqoop을 이용한 데이터베이스 통합

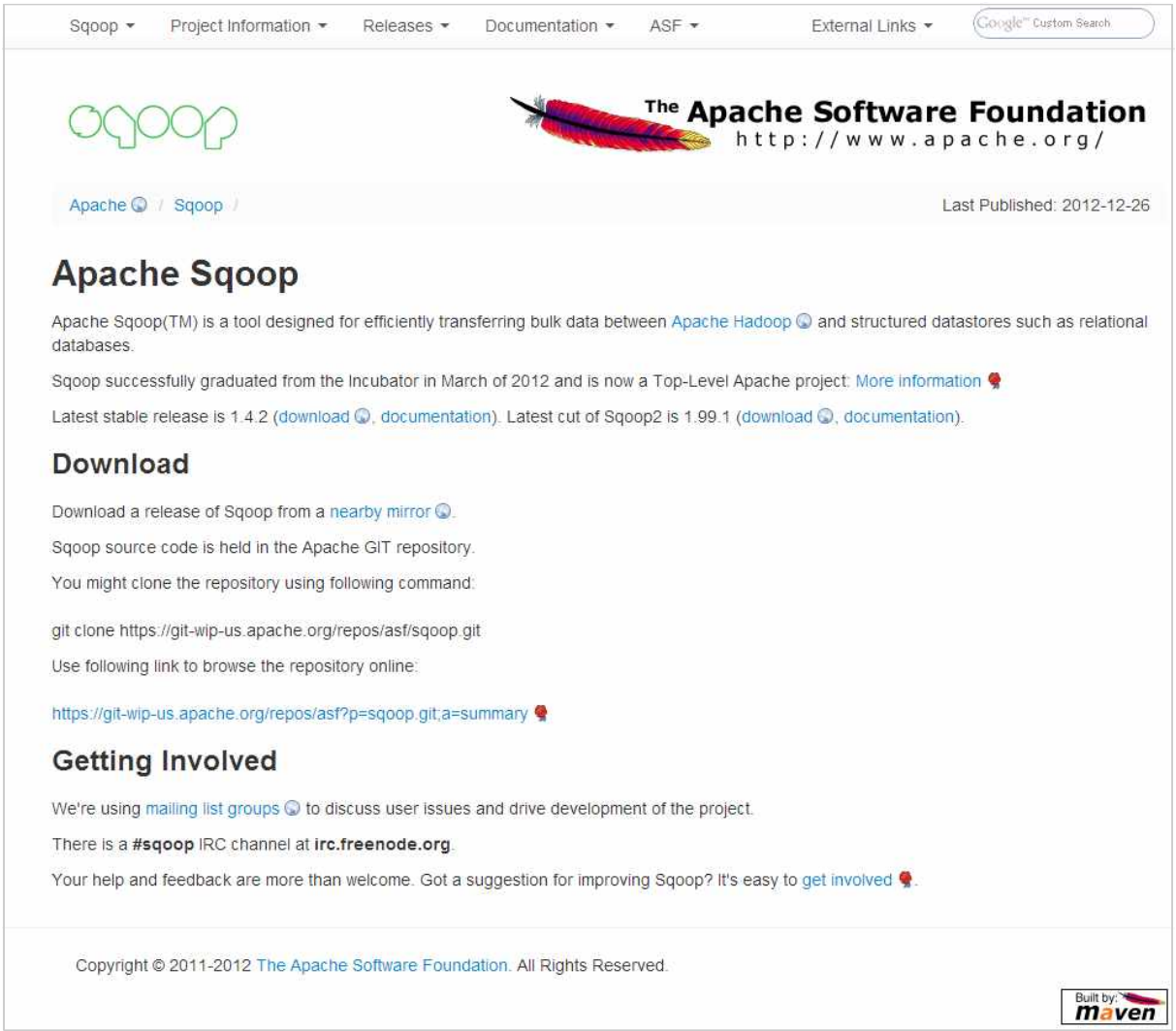
Apache Hadoop은 파일을 기반으로 모든 작업을 수행하고 관리하지만 서비스를 제공하기 위해서는 RDBMS나 NoSQL 등의 데이터 스토어에 데이터를 밀어 넣거나(Egress) 반대로 데이터 스토어에서 HDFS로 데이터를 저장(ingress)하는 다양한 작업이 필요하다.



다음의 도식을 보면 Apache Hadoop의 HDFS를 기반으로 MySQL에 데이터를 밀어 넣거나 꺼내기도 하고 필요한 경우 NoSQL인 MongoDB에 데이터를 밀어 넣고 꺼내는 작업을 수시로 진행하는 것을 확인할 수 있다.



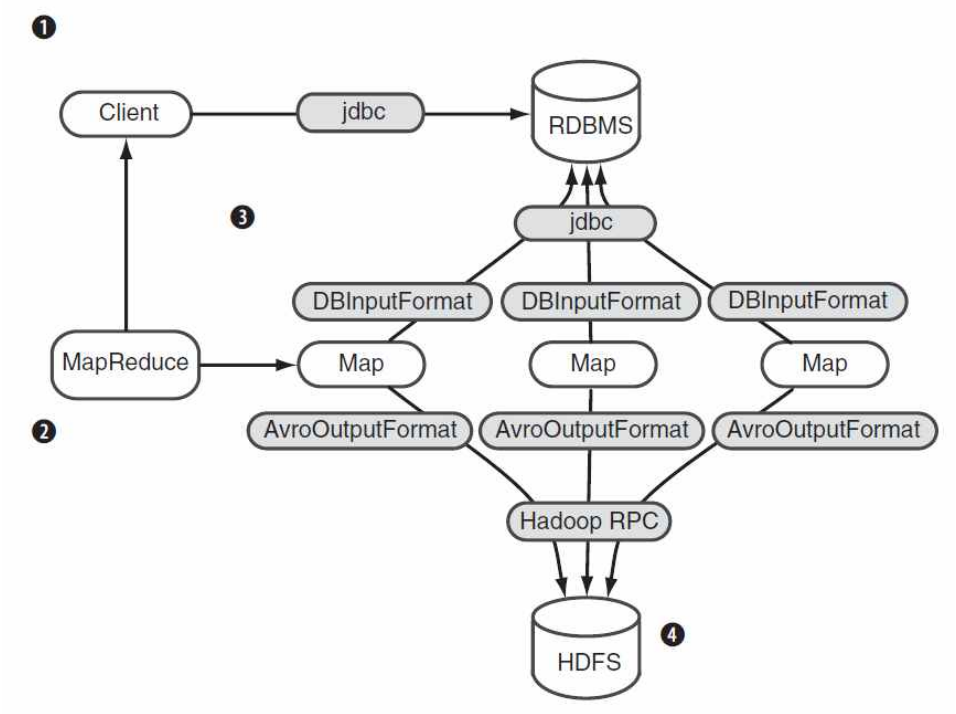
통상적으로 RDBMS와 Hadoop의 HDFS가 서로 데이터를 주고받을 때에는 Apache Incubator 프로젝트인 Sqoop(http://sqoop.apache.org)을 이용한다.



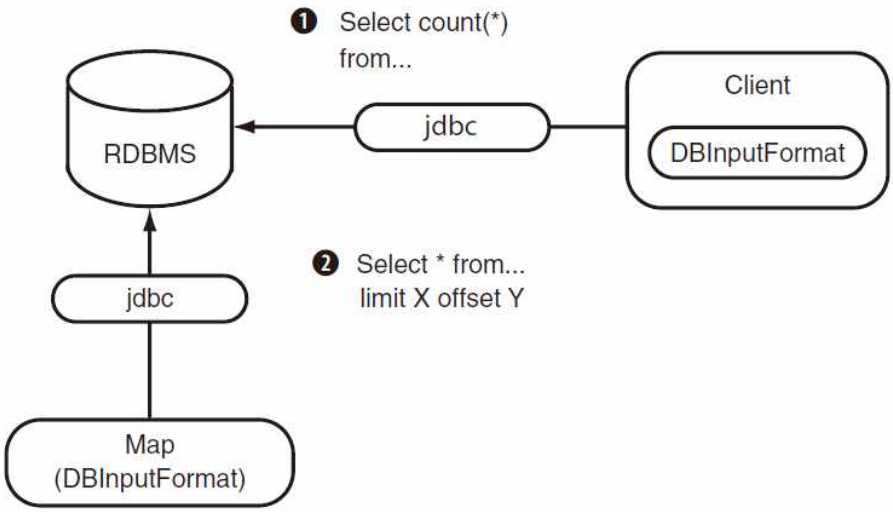
NoSQL과 Hadoop의 HDFS 간 데이터 교환은 Map와 NoSQL Client API를 이용하여 직접 구현하거나 또는 NoSQL에서 제공하는 Input Format을 사용한다.

1. Ingress

Ingress는 RDBMS 등의 데이터 스토어에 있는 데이터를 HDFS에 파일로 기록하는 작업이다. 다음은 RDBMS와 연동하여 Ingress를 수행하는 과정을 설명하는 도식으로 우선 ❶ 데이터베이스의 데이터를 HDFS로 밀어 넣기 위해서 데이터의 개수를 확인하고 ❷ RDBMS에 접속하여 데이터를 로딩하고 HDFS에 기록하는 MapReduce Job을 실행한다. ❸ 그리고 데이터베이스에서 데이터를 가져와서 ❹ HDFS에 파일로 기록하는 순서대로 Ingress를 처리한다.

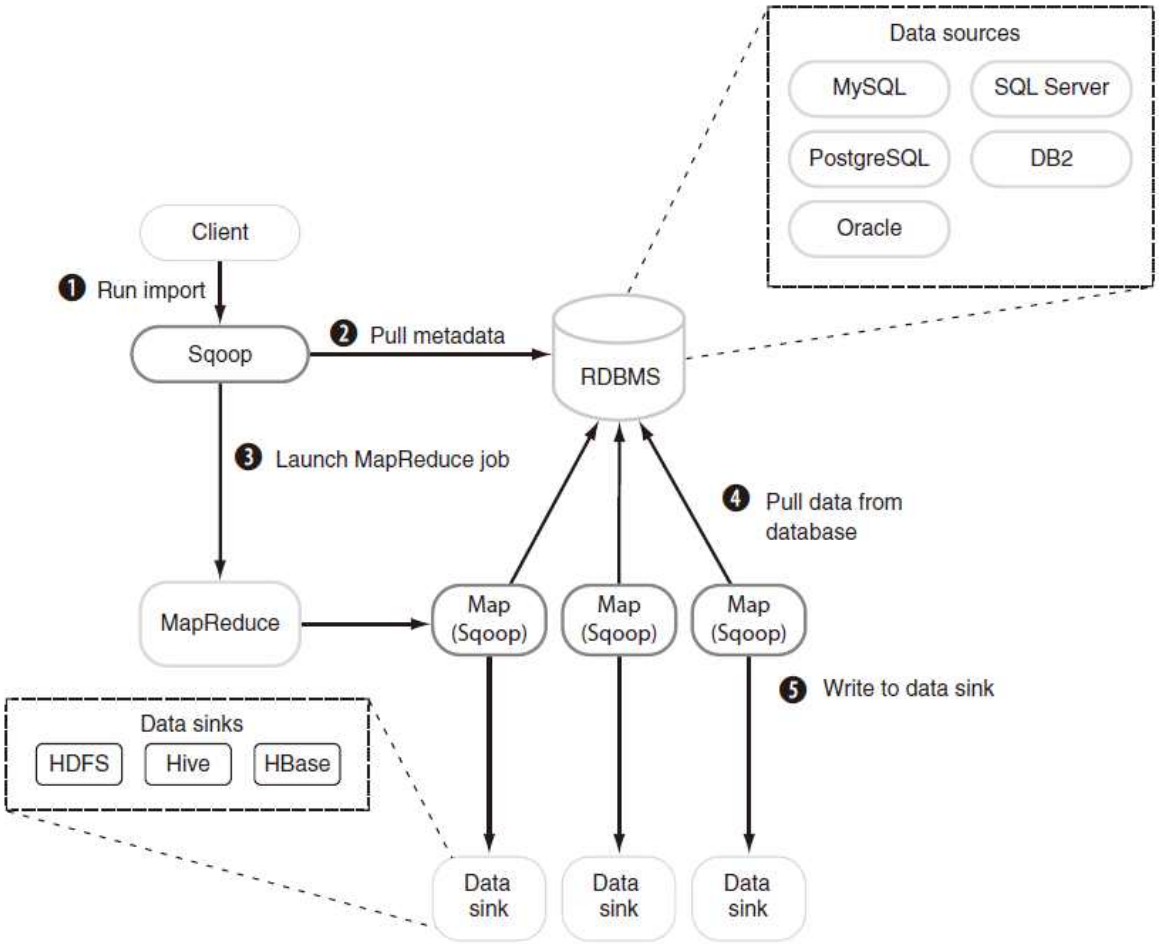


보통 이 작업을 제대로 수행하려면 ❶ Sqoop의 DBInputFormat이 로딩할 테이블의 총 건수를 계산하고 ❷ Map의 개수로 전체 개수를 나누어서 Offset을 결정해야만 각각의 Map이 해당 범위의 ROW를 Select하여 파일로 기록할 수 있게 된다.

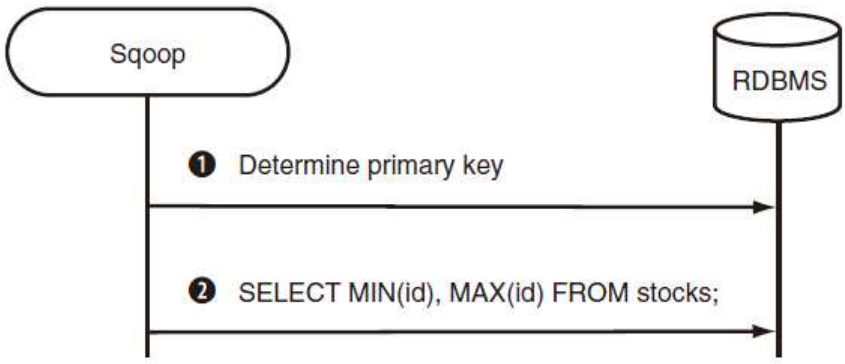


각각의 Map에서 로딩할 ROW의 개수와 범위가 결정되면 Sqoop은 다음과 같이 ❶ Sqoop Client가 동작하고 ❷ 데이터베이스의 메타 데이터를 수집한다. ❸ 그리고 나서 실제로 Map에서 RDBMS에 접근하여 데이터를 꺼내서 HDFS에 저장할 MapReduce Job을 실행한다. ❹

그러면 데이터베이스에서 실제로 데이터를 가져와서 ❺ HDFS의 파일에 기록한다.

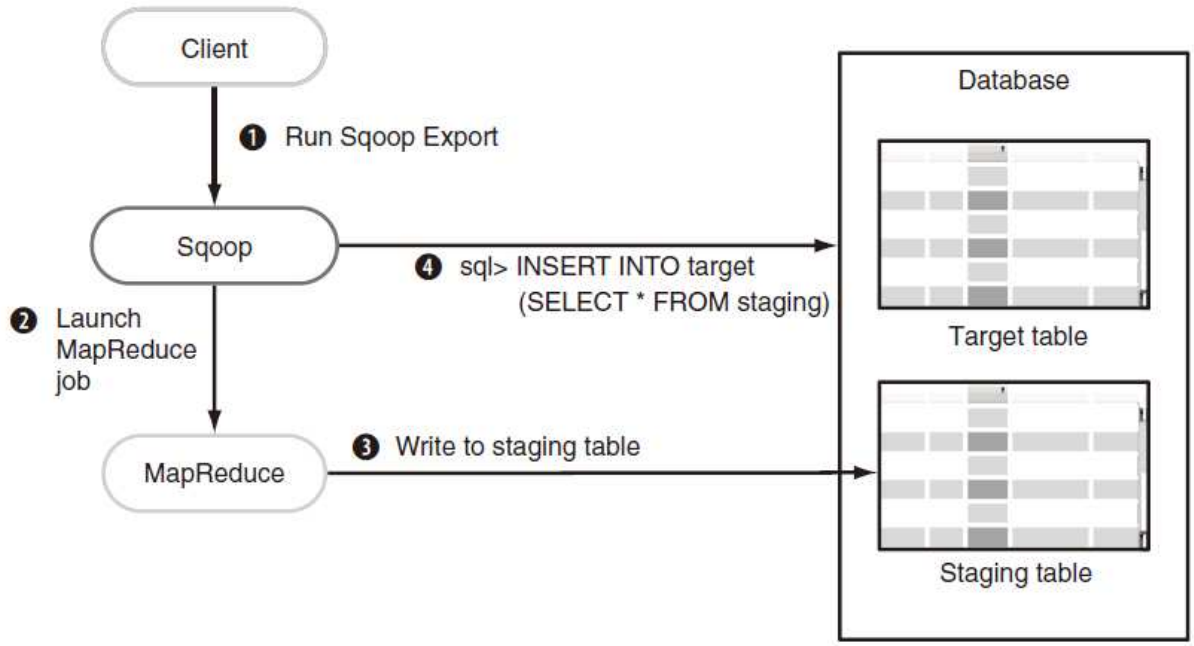


정말 중요한 작업은 Map에 할당을 해줄 Primary Key의 범위를 결정하는 것으로써 Sqoop은 다음과 같이 SQL Select를 이용하여 min, max를 계산하여 각 Map이 처리할 Primary Key의 범위를 결정한다.



2. Egress

데이터베이스의 데이터를 HDFS로 적재했다면 반대로 HDFS의 파일을 서비스를 위해서 RDBMS에 적재해야 한다. 보통 이 경우 Sqoop의 Export를 사용하여 다음과 같이 수행할 수 있다. 우선 ❶ Sqoop Export를 개발자가 실행하면 ❷ HDFS 파일의 내용을 데이터베이스 테이블로 Insert하는 MapReduce Job을 실행한다. ❸ 그러면 각 Map은 파일의 1 ROW를 데이터베이스의 스테이징 테이블에 먼저 Insert하고 모든 작업이 완료되면 ❹ Sqoop은 스테이징 테이블에 있는 내용으로 다시 최종 목적지 테이블에 기록한다.



3. Sqoop 사용시 주의할 점

Sqoop은 ingress, Egress 모두 데이터베이스에 직접 커넥션을 열어서 Map이 수행하는 구조로 되어 있기 때문에 ingress의 경우 HDFS의 파일로 기록할 데이터가 많은 경우 데이터베이스의 리소스를 장시간 점유하여 부하를 발생시키는 문제가 발생한다. 반대로 Egress의 경우 데이터베이스에 데이터를 insert하기 때문에 대량의 로그 파일을 insert하는 경우 데이터베이스에 심각한 영향을 줄 수 있다. 따라서 Sqoop을 사용하는 경우 동시에 데이터베이스에 접근하는 Map의 개수를 반드시 제한해야 한다. 보통 Map의 개수는 커맨드 라인 옵션을 통해서

다음과 같이 제어할 수 있다. Sqoop에 대한 상세한 사항은 (http://sqoop.apache.org/docs/1.4.2/SqoopUserGuide.html)을 참고 하도록 한다.

다음은 Sqoop의 커맨드 라인 옵션으로 -m 또는 --num-mappers 옵션을 이용하면 데이터베이스에 접근하는 Map의 개수를 제어할 수 있다. 현장에서 이 옵션을 이용하여 처리하는 경우 가장 부하를 유발하지 않고 최상의 성능을 내는 수치를 찾는 것이 중요하다. 이 값이 너무 크면 데이터베이스에 많은 부하가 발생하고, 이 값이 너무 작으면 너무 오랜 시간 동안 데이터를 처리하게 된다.

Sqoop 인자	설명
--append	Append data to an existing dataset in HDFS
--as-avrodatafile	Imports data to Avro Data Files
--as-sequencefile	Imports data to SequenceFiles
--as-textfile	Imports data as plain text (default)
--boundary-query 〈statement〉	Boundary query to use for creating splits
--columns 〈col,col,col…〉	Columns to import from table
--direct	Use direct import fast path
--direct-split-size 〈n〉	Split the input stream every n bytes when importing in direct mode
--inline-lob-limit 〈n〉	Set the maximum size for an inline LOB
-m,--num-mappers 〈n〉	Use n map tasks to import in parallel
-e,--query 〈statement〉	Import the results of statement.
--split-by 〈column-name〉	Column of the table used to split work units
--table 〈table-name〉	Table to read
--target-dir 〈dir〉	HDFS destination dir
--warehouse-dir 〈dir〉	HDFS parent for table destination
--where 〈where clause〉	WHERE clause to use during import
-z,--compress	Enable compression
--compression-codec 〈c〉	Use Hadoop codec (default gzip)
--null-string 〈null-string〉	The string to be written for a null value for string columns
--null-non-string 〈null-string〉	The string to be written for a null value for non-string columns

제3장 데이터 분석 및 처리

제1절 데이터 웨어하우스

Apache Hadoop EcoSystem 중에서 Apache Hive는 Data Warehouse를 구현하기 위한 오픈소스로써 데이터를 HDFS 상에 적제하고 이를 처리하는데 있어서 가장 편리한 사용자 언어인 SQL(Hive QL)을 제공한다.

1. 구성 요소

Apache Hadoop은 HDFS에 파일을 읽고 쓰지만 Hive는 SQL like Hive QL을 사용하기 때문에 SQL에서 파일을 직접 사용할 수 없다. 이러한 이유로 Hive는 HDFS 상의 파일과 테이블과 컬럼에 대한 메타 데이터 정보를 관리하는 메타데이터 스토어로 구성되어 있다. 사용자는 테이블을 생성하고 HDFS 상에 있는 파일을

2. Hive QL

가. DDL Operation

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
      page_url STRING, referrer_url STRING,
      ip STRING COMMENT 'IP Address of the User')
 COMMENT 'This is the page view table'
 PARTITIONED BY(dt STRING, country STRING)
 CLUSTERED BY(userid) SORTED BY(viewTime) INTO 32 BUCKETS
 ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\001'
   COLLECTION ITEMS TERMINATED BY '\002'
   MAP KEYS TERMINATED BY '\003'
 STORED AS SEQUENCEFILE;
```

나. Drop Table & Partition

Hive QL에서는 이미 생성한 테이블 및 파티션은 SQL과 동일하게 DROP TABLE 또는 DROP PARTITION을 이용하여 수정할 수 있다.

```
DROP TABLE pv_users;

ALTER TABLE pv_users DROP PARTITION (ds='2008-08-08')
```

다. 데이터 로딩

테이블을 생성하게 되면 우선적으로 HDFS 또는 로컬 파일 시스템의 파일을 테이블로 로딩해야 한다.

```
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/pv_gender_sum'
SELECT pv_gender_sum.*
FROM pv_gender_sum;
```

라. Join

Join이 비록 MapReduce에서 가장 시간이 많이 소요되는 작업이긴 하지만 특정 테이블의 데이터가 작은 경우 Map Side Join을 수행하면 높은 수준의 성능을 기대할 수 있다. 만약

```
SELECT a.* FROM a JOIN b ON (a.id = b.id AND a.department = b.department)

SELECT a.val, b.val, c.val
FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key2)

SELECT a.val, b.val, c.val
FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key2)
```

Hive QL은 LEFT, RIGHT, SEMI LEFT 등의 Join을 제공한다. 일반적으로 이러한 기능을 MapReduce 프로그래밍 하는 경우 경우에 구현의 난이도가 높아지는데 주로 발생하는 구현의 난이도는 하나 이상의 스키마로 구성된 다수의 경로를 입력 경로로 지정했을 때 각 Map에 할당되는 스키마가 달라서 Reduce로 동일한 Key로 전달될 때 순서가 변경되어 어느 파일의 컬럼이 먼저 앞에 와야 하는지 이해할 수 없는 경우와 Join할 Key가 일부 파일에만 존재하지 않는 경우, 그리고 이 두 가지 경우가 복합적으로 작용하는 경우이다. Hive QL은 그런 면에서 상당히 강력한 도구라 할 수 있으며 대부분의 경우 최적화 하지 않은 MapReduce 기반 Join보다 Hive QL 기반 Join이 훨씬 더 좋은 성능을 보여준다.

```
SELECT a.val, b.val FROM a LEFT OUTER JOIN b ON (a.key=b.key)

SELECT a.key, a.val FROM a LEFT SEMI JOIN b on (a.key = b.key)
```

Join이 비록 MapReduce에서 가장 시간이 많이 소요되는 작업이긴 하지만 특정 테이블의 데이터가 작은 경우 Map Side Join을 수행하면 높은 수준의 성능을 기대할 수 있다. 만약 한 쪽 테이블의 값이 매우 작다면 MapReduce 관점에서 봤을 때 이 파일을 각각의 Map에서 로딩하여 메모리 상에서 유지하다가 Join할 대상 파일을 로딩할 때 메모리 상에서 Join 컬럼을 찾아나가면 되므로 상대적으로 고성능을 수행할 수 있다. Join이 느린 핵심 이유는 Reduce로 데이터를 전달하기 때문이므로 이 과정을 생략한다면 높은 수준의 성능을 기대할 수 있다.

```
set hive.optimize.bucketmapjoin = true

SELECT /*+ MAPJOIN(b) */ a.key, a.value
FROM a join b on a.key = b.key
```

마. 집계 함수

Hive QL을 사용하는 가능 큰 이유 중에 하나는 바로 집계 함수를 사용하는 것이지만 현재 지원하는 Hive QL의 집계 함수는 상용 RDBMS에 비하여 턱없이 부족한 실정이다. 그 이유는 MapReduce의 처리 방식에서 기인한다. MapReduce의 처리 방식이 단일 노드의 처리 방식과 달라서 구현이 어렵기 때문에 현재 Hive QL에서는 제한적으로 지원하는 상황이다.

```
INSERT OVERWRITE TABLE pv_gender_agg
SELECT pv_users.gender, count(DISTINCT pv_users.userid),
        count(*), sum(DISTINCT pv_users.userid)
FROM pv_users
GROUP BY pv_users.gender;
```

다음은 Hive에서 제공하는 내장 집계 함수이다. RDBMS의 SQL을 Hive QL로 변환 시 집계 함수를 사용하는 경우 반드시 다음의 집계 함수를 확인하도록 한다.

반환형	집계 함수	설명
bigint	count(*) count(expr) count(DISTINCT expr[, expr_.])	count(*) - Returns the total number of retrieved rows, including rows containing NULL values; count(expr) - Returns the number of rows for which the supplied expression is non-NULL; count(DISTINCT expr[, expr]) - Returns the number of rows for

		which the supplied expression(s) are unique and non-NULL.
double	sum(col), sum(DISTINCT col)	Returns the sum of the elements in the group or the sum of the distinct values of the column in the group
double	avg(col), avg(DISTINCT col)	Returns the average of the elements in the group or the average of the distinct values of the column in the group
double	min(col)	Returns the minimum of the column in the group
double	max(col)	Returns the maximum value of the column in the group
double	variance(col), var_pop(col)	Returns the variance of a numeric column in the group
double	var_samp(col)	Returns the unbiased sample variance of a numeric column in the group
double	stddev_pop(col)	Returns the standard deviation of a numeric column in the group
double	stddev_samp(col)	Returns the unbiased sample standard deviation of a numeric column in the group
double	covar_pop(col1, col2)	Returns the population covariance of a pair of numeric columns in the group

double	covar_samp(col1, col2)	Returns the sample covariance of a pair of a numeric columns in the group
double	corr(col1, col2)	Returns the Pearson coefficient of correlation of a pair of a numeric columns in the group
double	percentile(BIGINT col, p)	Returns the exact pth percentile of a column in the group (does not work with floating point types). p must be between 0 and 1. NOTE: A true percentile can only be computed for integer values. Use PERCENTILE_APPROX if your input is non-integral.
array<double>	percentile(BIGINT col, array(p1 [, p2]...))	Returns the exact percentiles p1, p2, ... of a column in the group (does not work with floating point types). pi must be between 0 and 1. NOTE: A true percentile can only be computed for integer values. Use PERCENTILE_APPROX if your input is non-integral.
double	percentile_approx(DOUBLE col, p [, B])	Returns an approximate pth percentile of a numeric column (including floating point types) in the group. The B parameter controls approximation accuracy at

		the cost of memory. Higher values yield better approximations, and the default is 10,000. When the number of distinct values in col is smaller than B, this gives an exact percentile value.
array<double>	percentile_approx(DOUBLE col, array(p1 [, p2]...) [, B])	Same as above, but accepts and returns an array of percentile values instead of a single one.
array<struct {'x','y'}>	histogram_numeric(col, b)	Computes a histogram of a numeric column in the group using b non-uniformly spaced bins. The output is an array of size b of double-valued (x,y) coordinates that represent the bin centers and heights
array	collect_set(col)	Returns a set of objects with duplicate elements eliminated

바. Union

Union은 RDBMS와 사용법이 다르지 않으며 Hive QL에서는 다음과 같이 FROM 내부에서 사용할 수 있다. 보통 Union은 MapReduce 관점에서 Map 작업을 통해서 수행하므로 빠른 결과를 수행하므로 Hive QL에서 이러한 관점에서 Job의 Map/Reduce의 개수를 모니터링 하여 최적화 하도록 한다. 일반적으로 Union은 MapReduce에서 두 번의 Map 또는 단일 Map을 통해서 커스터마이징 할 수 있다.

```
INSERT OVERWRITE TABLE actions_users
SELECT u.id, actions.date
FROM (
    SELECT av.uid AS uid
    FROM action_video av
    WHERE av.date = '2008-06-03'
  UNION ALL
    SELECT ac.uid AS uid
    FROM action_comment ac
    WHERE ac.date = '2008-06-03'
) actions JOIN users u ON(u.id = actions.uid);
```

사. Sub Query

Hive QL에서 Sub Query는 RDBMS에 상대적으로 제한적이긴 하지만 다음의 수준에서 사용할 수 있다.

```
SELECT col
FROM (
  SELECT a+b AS col
  FROM t1
) t2
```

필요하다면 다음과 같이 UNION ALL과 같이 사용하여 하나 이상의 테이블을 Sub Query에서 사용할 수 있다.

```
SELECT t3.col
FROM (
  SELECT a+b AS col
  FROM t1
  UNION ALL
  SELECT c+d AS col
  FROM t2
) t3
```

3. 복잡한 구조를 가진 파일 처리

입력 파일이 1차원 구조를 가진 단순한 파일도 있지만 복잡한 구조를 가진 파일도 현장에서는 매우 많이 목격한다. 다음의 로그 파일은 그 구조를 파악하는 것도 어려울 만큼 복잡한 구조로 되어 있다. 한 줄로 구성된 로그 파일이지만 다양한 구분자로 구성된 꽤나 복잡한 자료 구조를 가지고 있다. 이러한 자료구조를 표현하는 스키마를 구성하는 것은 굉장히 어려운 일이다.

```
John Doe^A100000.0^AMary Smith^BTodd Jones^AFederal Taxes^C.2^BState Taxes^C.
05^BInsurance^C.1^A1 Michigan Ave.^BChicago^BIL^B60600
```

이러한 구조를 가지고 있는 파일을 Hive에서 다루려면 이에 맞는 테이블을 생성해야 한다. 다음은 위 파일의 구조를 Hive의 스키마 구조로 표현한 것이다.

```
CREATE TABLE employees (
  name STRING,
  salary FLOAT,
  subordinates ARRAY<STRING>,
  deductions MAP<STRING, FLOAT>,
  address STRUCT<street:STRING,
  city:STRING, state:STRING, zip:INT>
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\001'                    → ^A (필드 구분자)
COLLECTION ITEMS TERMINATED BY '\002'          → ^B (배열/구조체/Map간 구분자)
MAP KEYS TERMINATED BY '\003'                  → ^C (Map의 KeyValue 구분자)
LINES TERMINATED BY '\n'                       → 라인 구분자
STORED AS TEXTFILE;
```

4. 데이터 파티셔닝

데이터를 다루다 보면 관리 및 성능을 위해서 데이터를 특정 기준(예; 날짜)으로 구분하여 사용하는 경우가 있다. 보통 이러한 기법을 파티셔닝이라고 한다. 파티셔닝을 하는 이유는 크게 ▲ 쿼리의 성능 향상 ▲ 데이터 구조화이다. 대용량 데이터가 파일 시스템의 한 디렉터리에서 모두 관리하고 있다면 불필요한 데이터를 삭제하거나, 검색하고자 하는 경우 전체 데이터를 다 읽어야 하는 경우가 발생한다. 이러한 문제를 해결하기 위해서 검색 기준으로 데이터를 파티셔닝할 수 있다. 예를 들어 날짜(년, 월, 일)를 기준으로 조회한다면 Hive 내에서는 로그 파일을 보관할 때 디렉터리를 년, 월, 일을 기준으로 정리해야 한다. 다음은 검색 기준에 따라서 Hive에서 테이블을 생성할 때 년, 월, 일을 기준으로 파티셔닝을 한 경우이다.

```
CREATE TABLE message_log (status STRING, msg STRING, hms STRING)
       PARTITIONED BY (year INT, month INT, day INT);
```

이렇게 파티셔닝한 로그 파일은 HDFS 상에서 다음과 같이 보관하고 있어야 한다.

```
message_log/year=2011/month=12/day=31/
message_log/year=2012/month=01/day=01/
...
message_log/year=2012/month=01/day=31/
message_log/year=2012/month=02/day=01/
```

실제 데이터를 조회하는 경우 다음과 같이 WHERE에 파티셔닝 정보를 기준으로 조건을 지정하게 되면 최소의 파일을 로딩하여 처리하게 된다. 만약 SELECT * FROM message_log 쿼리를 직접 실행하게 되면 전체 파일을 모두 읽어서 처리하는데 매우 오랜 시간이 소요된다.

```
SELECT * FROM message_log
WHERE year = 2012 AND
      month = 01 AND
      day = 31;
```

5. SerDe(Serializer/Deserializer)

Hive의 SerDe는 복잡한 로그 파일을 테이블에 import할 때 로그 메시지를 수동으로 제어할 수 있도록 하는 기능이다. 예를 들어 복잡한 Apache HTTP Server의 로그 메시지를 Hive의 테이블로 import하는 경우 로그 메시지의 파싱 작업이 먼저 일어나야 한다. 이 경우 보통 Hadoop의 MapReduce 프로그래밍을 이용하여 선처리 하지만 SerDe를 이용하면 제한적으로 이러한 작업을 MapReduce 프로그래밍을 하지 않고도 가능하게 할 수 있다. 다음은 Apache HTTP Server의 로그 파일을 Regular Expression을 이용하여 한 번에 처리하는 예제이다.

```
add jar ../build/contrib/hive_contrib.jar;

CREATE TABLE apachelog (
  host STRING,
  identity STRING,
  user STRING,
  time STRING,
```

```
  request STRING,
  status STRING,
  size STRING,
  referer STRING,
  agent STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
  "input.regex" = "([^]*) ([^]*) ([^]*) (-|\\[^\\]*\\]) ([^ \"]*|\"[^\"]*\")
   (-|[0-9]*) (-|[0-9]*)(?: ([^ \"]*|\".*\") ([^ \"]*|\".*\"))?",
   "output.format.string" = "%1$s %2$s %3$s %4$s %5$s %6$s %7$s %8$s %9$s"
)
STORED AS TEXTFILE;
```

** https://cwiki.apache.org/confluence/display/Hive/SerDe

6. 파티션 필터

데이터베이스에서 대용량 데이터를 조회하는 경우(작은 데이터를 조회하더라도) 필요한 데이터만 사용하는 것이 바람직하다는 것은 누구나 다 이해하는 사실이다. 예를 들면 stocks 테이블의 전체 컬럼이 100개라 할지라도 실제로 필요한 컬럼이 2개면 2개만 Select하는 것이 바람직하다.

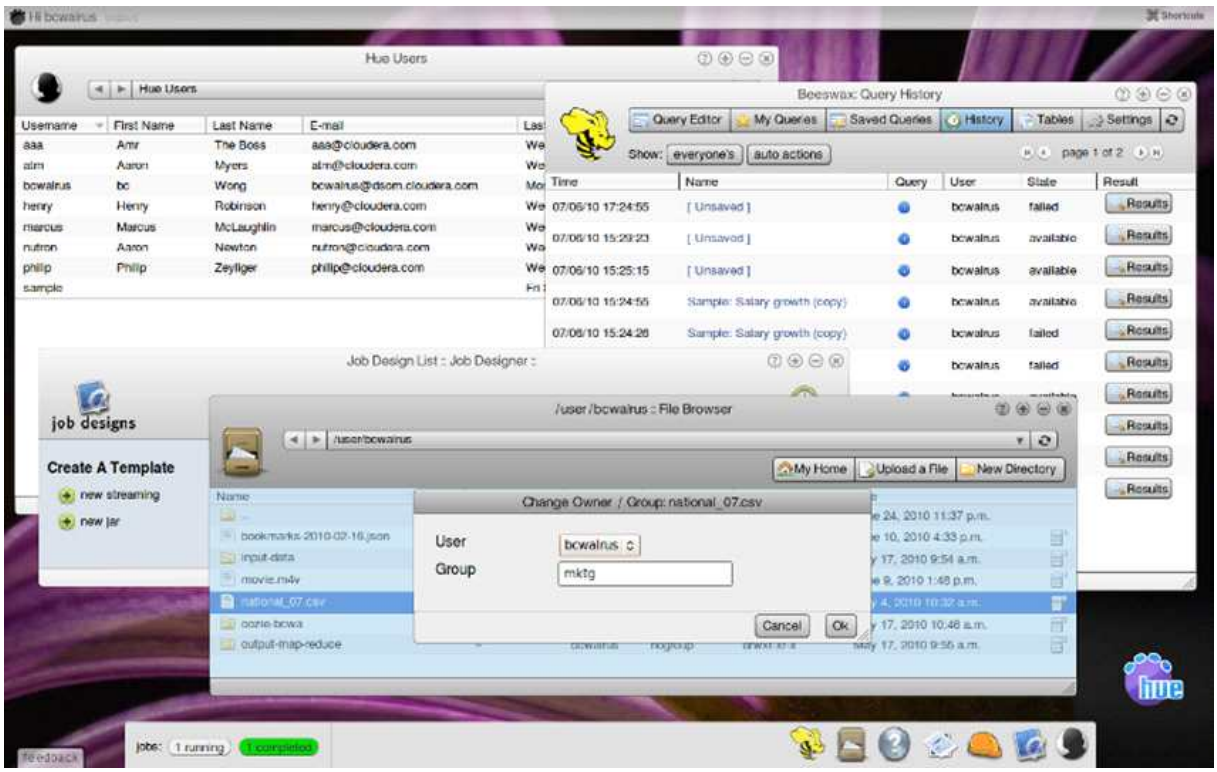
```
SELECT ymd, symbol FROM stocks
WHERE exchange = 'NASDAQ' AND symbol = 'AAPL' ;
```

하지만 Hadoop에서 만큼은 예외다. 위와 같이 Hive QL을 작성한 경우 실제로 조회시 MapReduce Job이 1회 더 수행하게 된다. 그 이유는 2개의 컬럼을 추출하기 위해서이다. 하지만 다음과 같이 모든 컬럼을 저장하게 되면 별도의 MapReduce Job이 동작하지 않아서 성능에 이득이 발생한다.

```
SELECT * FROM stocks
WHERE exchange = 'NASDAQ' AND symbol = 'AAPL' ;
```

7. HUE(Hive 웹 관리 도구)

Apache Hive가 매우 강력한 도구임에 틀림없지만 여전히 커맨드 라인 방식의 개발 방법은 생산성 및 접근성 면에서 부족한 점이 많다. 이러한 문제를 극복하기 위한 방법이 오픈소스인 HUE를 사용하는 것으로 HUE는 웹 브라우저를 이용하여 Hive를 사용할 수 있는 환경을 제공한다. Cloudera Hadoop 배포판에 기본으로 포함되어 있으므로 HUE를 사용하는 경우 Cloudera Hadoop 배포판인 CDH를 사용할 것을 권고한다.

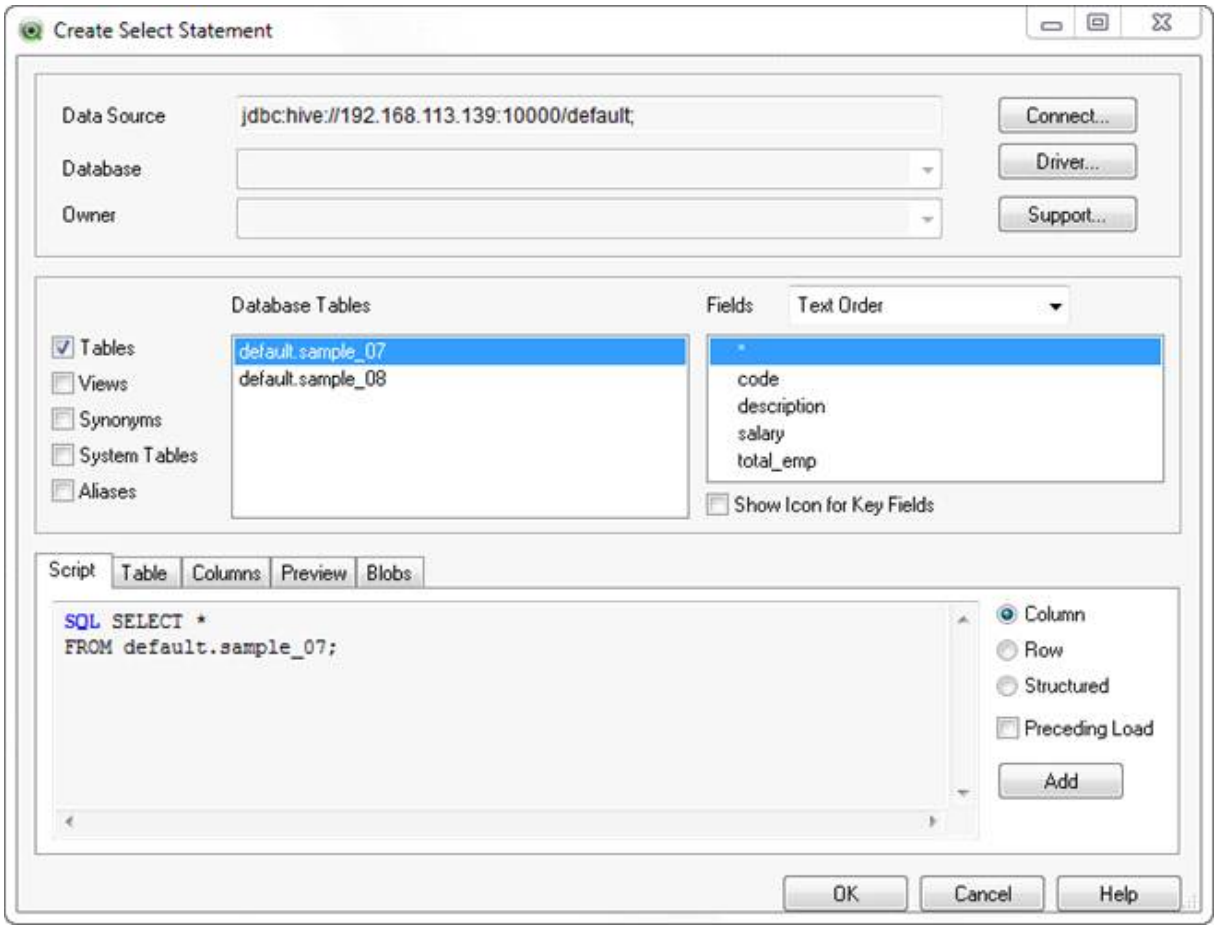


http://www.cloudera.com/blog/2010/07/whats-new-in-cdh3b2-hue

8. JDBC Connector

과거 DW를 수행했던 대부분의 사람들은 편리한 사용자 인터페이스를 가진 상용 도구를 이용하였다. 때문에 Hive의 커맨드 라인은 이러한 사람들에게 매우 불편한 도구임이 틀림없

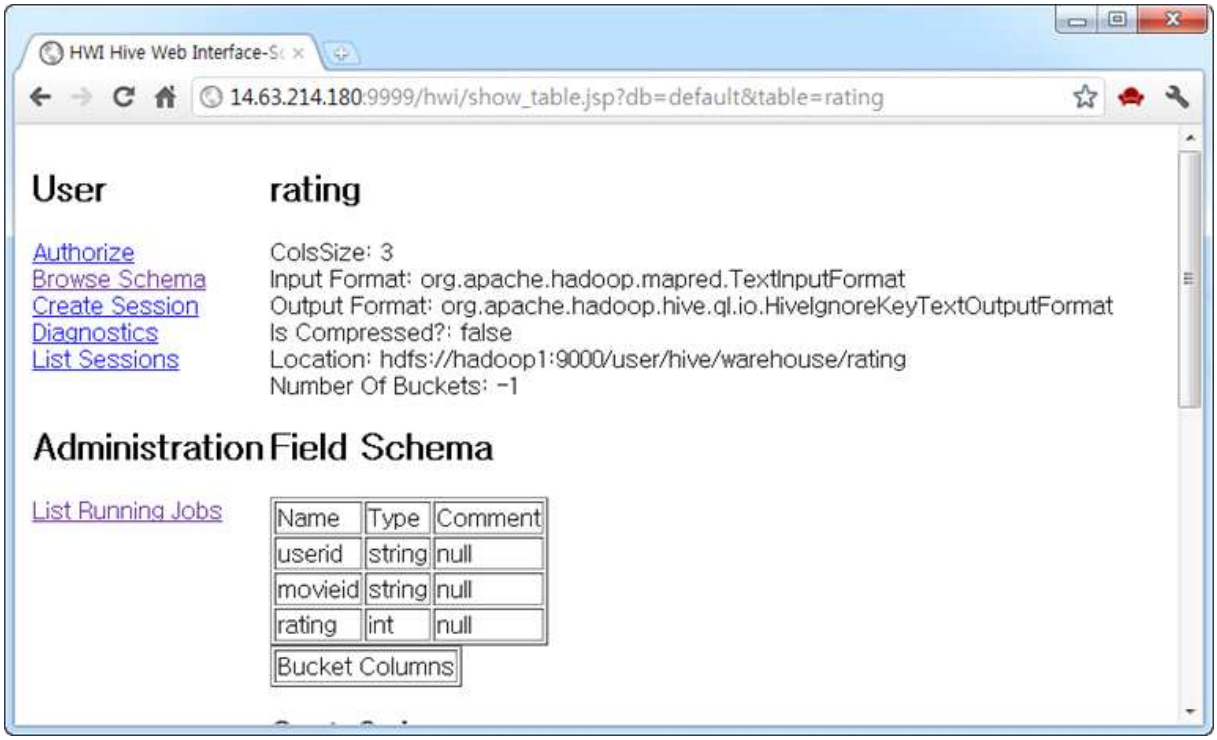
다. 이러한 문제를 극복하기 위해서 Hive를 표준 인터페이스인 JDBC로 연결하는 도구를 활용하면 다음과 같이 JDBC를 지원하는 기존 도구를 이용하여 Hive QL을 수행할 수 있다.



http://community.qlikview.com/docs/DOC-2457

9. Hive 관리 화면

Hive 자체도 별도의 관리 화면을 제공한다. 스키마 정보와 진단 정보 등을 관리하는 화면은 별도로 사용할 수 있지만 기본으로는 동작하지 않는다. 이 화면을 조직 내부에서 사용하는 관리 시스템의 일부 화면으로 포함하고자 하는 경우 #<HIVE_HOME>/bin/hive --service hwi 커맨드를 이용하여 별도 서버를 구동하고 다음의 페이지에 접속하도록 한다.



10. 도입 시 주의사항

비록 Apache Hive가 SQL을 제공하여 사용자의 편리성을 강조하고 있지만 도입 시 다음을 먼저 고려해야 한다. 다음의 중요 포인트를 간과하고 RDBMS에서 사용하던 SQL을 Hive QL로 변환하는 경우 성능 및 기능 부족으로 변환 자체가 불가능할 수 있다.

- **빠른 응답을 가진 SQL Select 조회는 적합하지 않다.** Hive는 Hive QL을 MapReduce로 변환하므로 빠른 응답을 가진 결과를 얻기 보다는 대용량 배치 처리를 통해 결과를 얻는데 더 적합하다. 빠른 응답을 갖기를 원한다면 Impala를 사용하도록 한다. 다만 Impala는 아직 성숙도가 높지 않으므로 현장에서 사용하는 경우 Hadoop 버전부터 다시 고려해야 하므로 Hadoop 2.0 도입을 먼저 고려하도록 한다.
- **CRUD에서 CUD 작업에는 적합하지 않다.** 이 경우 NoSQL 또는 RDBMS를 사용하도록 한다.
- **Hive QL에서 지원하는 in-line function을 확인한다.** Oracle과 같은 상용 제품에서 지원하는 모든 기능을 제공하지 않으며 그 중 매우 제한된 기능만 제공하므로 도입 전 반드시 검토해야 한다.

- **파티셔닝 및 조회 조건에 따라서 디렉터리 구조를 변경한다.** Hive는 파티셔닝 및 조회 조건에 따라서 특수한 디렉터리 구조를 가지고 있으므로 이에 상응하여 고성능을 구현할 수 있도록 디렉터리 구조를 조정해야 한다.
- **Select의 컬럼의 개수를 *로 지정하여 조회한다.** 성능 요구사항에 민감하다면 Select를 통해 조회할 컬럼은 *를 이용하여 전체를 다 생성하도록 한다. *로 설정하지 않고 컬럼을 직접 지정한 경우 Hive는 map 단계를 추가로 수행한다. 입력 파일이 대용량 파일인 경우 이 map을 동작하는 시간이 추가로 더 소요될 수 있다. 물론 map 작업이 빠르긴 하나 성능 요구사항이 매우 중요하다면 고려할 수 있다. 다만 이 방법은 더 많은 양의 파일을 생성하므로 디스크 공간을 더 소비한다.

제2장 실시간 스트리밍

제1절 비즈니스 요구사항과 실시간

최근 나타나는 현상 중 하나로 비즈니스 요구사항에 실시간 요소가 포함되어 있다는 것이다. 예를 들면 차량에 센서를 붙여서 차량의 위치를 실시간 탐지하여 이동거리를 예측한다던가, 웹 사이트 및 서비스 내에서 사용자의 활동을 실시간으로 모니터링 하여 사용자 인터페이스를 실시간 개인화 한다든지 하는 요건을 제대로 구현하려면 실시간으로 처리하는 기술을 먼저 확보해야 한다.

하지만 대부분의 빅 데이터에서는 장시간 시간이 소요되는 배치 중(Apache Hadoop 중심)만 논의되는 경향이 있으나 실제 현장에서는 실시간 속성은 매우 중요한 요구사항이다. 빅 데이터의 주요 속성 중에 하나인 velocity는 데이터가 생성되는 속도, 데이터의 수집 속도, 데이터의 전달 속도에 따라서 처리를 달리해야 함을 의미하는 용어로 이 분야는 전통적으로 Complex Event Processing(CEP) 및 Event Driven Architecture(EDA)라는 기술 영역으로 구분하고 있으나 최근 실시간 빅 데이터 처리를 위한 기반 기술 중 하나로써 인정하는 추세이다.

용어	정의
이벤트	- 실제로 발생한 사건, 일, 메시지 - 상태의 변경 - 특정한 액션 또는 상태의 변화를 통해 발생하는 변경이 불가능한 과거의 기록
이벤트 스트림	- 시간의 순서대로 연속되는 이벤트의 흐름 - 시작과 끝이 없는 이벤트의 연속된 흐름

CEP 또는 EDA를 기반으로 하는 애플리케이션은 이벤트 중심의 rule 또는 특징 시간 동안 이벤트를 처리할 수 있다는 장점은 있지만 scale out이 어렵기 때문에 빅 데이터 영역에서는 이벤트 중심의 프로세싱 이외에도 scale out이 가능한 스트리밍 중심의 프로세싱을 사용한다. 기술적으로 분류하면 다음의 특징이 있다.

구분	기술적 특징
이벤트 중심 처리	- 시간에 따른 이벤트의 일련의 연속된 흐름을 처리 - 특정한 time window 또는 건수를 연속적으로 처리 - Scale up 아키텍처 - 선언적 rule 기반 처리 - 단순한 시스템 구성
분산 스트리밍 중심 처리	- 시간에 따른 이벤트의 일련의 연속된 흐름을 처리 - Scale out 아키텍처 - Computation 중심 처리 - 복잡한 시스템 구성

Scale up, scale out 기술을 활용한 CEP와 분산 스트리밍 기술은 기술적으로는 서로 상호 배타적인 관계이지만 현장에서는 상호 보완적인 관계이므로 현장에서 필요에 따라서 적절히 혼재하여 시스템을 구현한다. 문제는 비즈니스 요구사항 중에서 실시간 요구사항이 포함되면 전체 시스템의 복잡도는 자연스럽게 증가한다는 것이며 어떤 기술을 어떻게 사용할 것인가를 결정하는 일이다.

예를 들어 실시간으로 발생하는 사용자의 클릭 스트림을 이용하여 가장 사용자들이 많이 구매하거나 상품정보를 실시간으로 현황을 보고 싶다면 반드시 실시간 스트리밍 기술 또는 이벤트 기술을 사용해야 한다. 하지만 여기에 다음과 같은 요구사항이 추가되면 시스템 아키텍처는 기존에는 경험해보지 못한 수준으로 복잡해진다. 대부분의 요구사항이 비기능적(non-functional requirement) 요구사항이라는 것도 큰 특징이다.

- 특정한 시간 동안 발생하는 이벤트 스트림 데이터 또는 이벤트의 처리 여부
- 메시지 또는 이벤트 손실 여부
- 메시지 또는 이벤트의 일부 내용을 변환해야 하는 경우
- 서비스가 늘어날수록 빠르게 급격하게 늘어나는 이벤트 스트림을 수용해야 하는 경우
- 특정 시간대에 이벤트 스트림의 수가 평상시의 수십 배로 늘어나는 경우
- 이벤트 스트림 처리의 복잡도가 높은 경우
- 외부 데이터 소스와 연계해야 하는 경우
- 이벤트 스트림을 저장해서 보관해야 하는 경우

보통 실시간 요건에는 두 개 이상의 비기능적 요구사항을 구현해야 하므로 실시간 서비스 요건을 수용하는 것은 기술적으로 상당히 어려울 수밖에 없다. 본 문서에서는 현재 실제 운영 중이거나 또는 이미 검증된 기술을 이용하여 실시간 서비스 요건을 수용하는 참조 아키텍처를 소개하고자 한다.

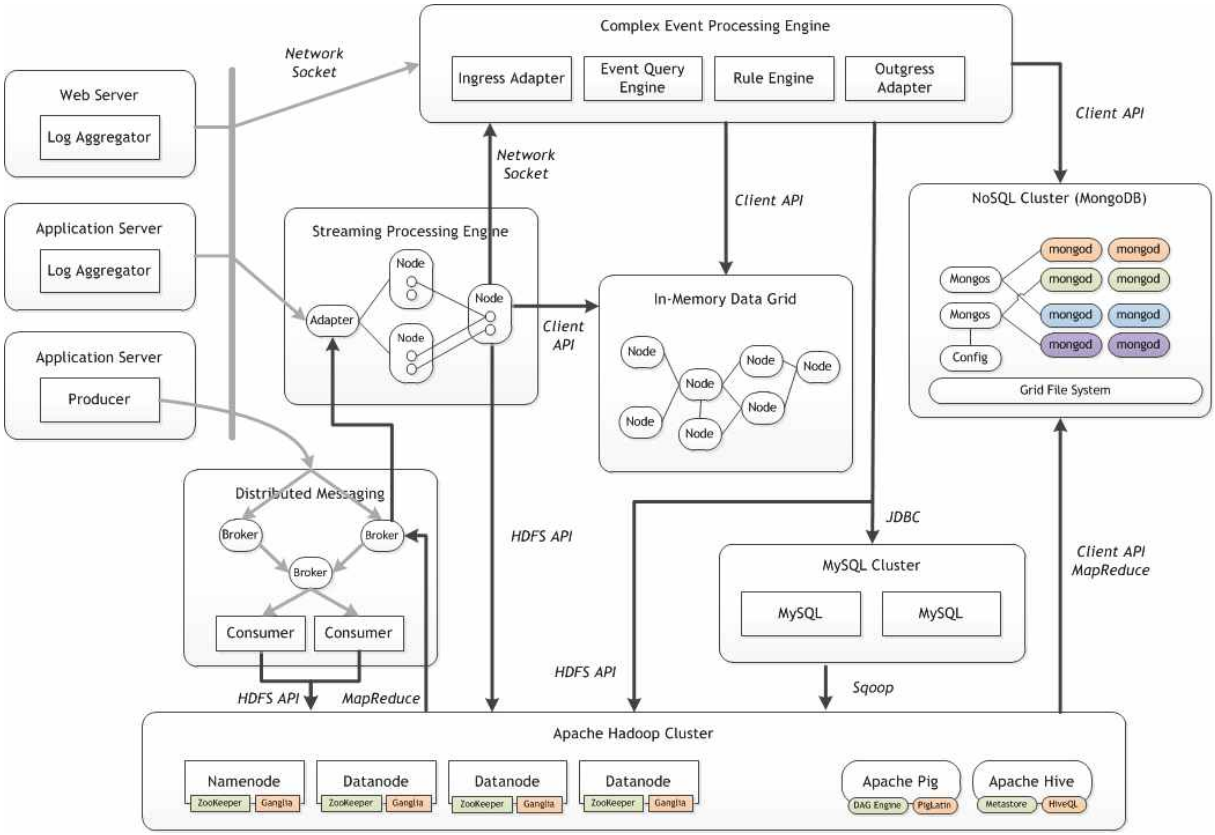
제2절 실시간(리얼타임) 처리 기술

현장에서 가장 어려운 문제 중 하나는 실시간 요구사항을 처리할 수 있는 기술과 시스템 아키텍처를 수립하는 것이다. 빅 데이터와 클라우드 기반 기술들은 지금까지 우리가 접했던 기술의 난이도와 아키텍처 그리고 복잡도 측면에서 그 격차가 매우 크다. 그러므로 우선적으로 실시간 처리를 위한 다양한 오픈소스를 기능적, 비기능적 특성을 먼저 이해해야 한다. 다음은 현재 실시간 빅 데이터 분야에서 가장 많이 사용하는 오픈소스이다.

오픈소스	라이센스	언어	확장 방법	즉시 Rule 추가 및 변경	필요한 인프라	Rule 유형
Esper CEP	GPL2 Commercial	Java	Scale up	예	없음	선언적 SQL Query Like
Drools Fusion CEP	ASL 2.0	Java	Scale up	예	없음	선언적 일반적은 Rule 기반 Query 지원
Storm	EPL 1.0	Clojure	Scale Out	Zookeeper 기반 구현 가능	ZeroMQ Zookeeper	프로그래밍
Apache S4	ASL 2.0	Java	Scale Out	Zookeeper 기반 구현 가능	Zookeeper (옵션) YARN (옵션)	프로그래밍
Apache Kafka	APL 2.0	Java	Scale Out		Zookeeper	프로그래밍

오픈소스	문서화	성숙도	커뮤니티	URL	참고
Esper	매우 좋음	높음, 안정적	중간	esper.codehaus.org	
Drools Fusion	좋음	3년 이상, 안정적	작음	www.jboss.org	
Storm	있음	운영에 사용	빠르게 성장	github.com/nathanmarz/storm	배포 기능 좋음
Apache S4	평균	낮음, 운영에 사용	중간	incubator.apache.org/s4	
Apache Kafka	좋음	운영에 사용	작음	incubator.apache.org/kafka	

각각의 오픈소스는 하나로 구성을 할 수도 있지만 통합하여 구성할 수도 있다. 이것을 결정하는 기준은 순전히 비즈니스 요구사항과 비기능적 요구사항에 달려있다. 다음은 이러한 다양한 오픈소스를 기준으로 합쳐진 통합 아키텍처이다.



각각의 서버 및 애플리케이션 서버에서 나오는 데이터는 그 용도에 따라서 Complex Event Processing(CEP) Engine으로 직접 전달되는 경우도 있고, 분산 메시징 시스템으로 전달되어 수집되는 경우도 있다. CEP로 전달된 이벤트는 특정한 시간동안 유지하면서 해당 시간 동안(time window)에 조건에 맞는 이벤트가 수집이 되면 수집된 이벤트를 별도의 NoSQL, RDBMS, Hadoop의 HDFS 등에 적재한다. 보통 CEP를 사용하는 경우 서비스 요구사항에는 "특정한 시간 동안" 이라는 요구사항이 포함되어 있는 경우가 많다. 예를 들어 "최근 10분 동안 시청에 있었던 20대 대학생" 이라는 요구사항이 포함되어 있으면 10분은 CEP를 사용해야 하는 조건이고 20대 대학생은 개인 정보 또는 추정 정보를 활용해야 하는 조건이다. 그러므로 전자는 CEP, 후자는 NoSQL, RDBMS 등과 연계를 해야 한다.

분산 메시징 시스템은 안정적으로 실시간 데이터 수집을 책임진다. 분산 메시징 시스템의 클라이언트는 서버 또는 서비스에 배포되고 메시지를 송신하는 주체이므로 publisher라고 부른다. 이 publisher가 메시지를 분산 메시징 시스템으로 전달하게 되면 분산 메시징 시스템은 내부적으로 부하를 분산하여 특정한 목적지로 안전하게 메시지를 전달한다. 메시지를 수신한 목적지는 자신에게 등록되어 있는 consumer로 메시지를 실시간으로 비동기 처리를 통해 전달

한다. Consumer는 타 시스템으로 메시지를 전달할 수도 있고 Hadoop의 HDFS에 파일로 적재할 수도 있다. 일반적으로 분산 메시징 시스템은 메시지를 안전하게 전달하는 것이 목표이므로 고객 또는 서비스 요구사항 중에서 “메시지 손실을 최소화해야 한다”는 비기능적 요구사항이 포함되면 매우 좋은 선택이 될 수 있다.

CEP와 분산 메시징 시스템 이외에도 분산 스트리밍 기술도 있다. 이 스트리밍 기술은 분산 메시징 시스템과 비슷한 역할을 하지만 메시지를 중간에 변환하거나, 집계를 하는 등의 기능을 수행할 수 있다. Apache S4는 실시간 검색 엔진을 구현하기 위한 요건으로 개발한 오픈소스이며, STORM은 리얼타임 Hadoop이라는 별칭을 가지고 있는 오픈소스이다. Apache S4는 이벤트 스트림의 유형에 따라서 분산하여 집계 기능을 수행할 수 있는데 주요 사례로 트위터에서 가장 인기 있는 토픽의 상위 Top 10을 뽑아내는 것이다. 보통 이 경우 각 노드에 부하가 분산되어 토픽의 개수를 취합하게 된다.

이렇듯 실시간 요건에 따라서 어떻게 시스템을 구현하느냐는 각 기술을 어떻게 사용할 것인가로 요약된다. 그러므로 각 오픈소스의 동작 특성과 기술 요소를 이해하는 것이 매우 중요하다 할 수 있다.

제3절 리얼타임 스트리밍 기반 아키텍처

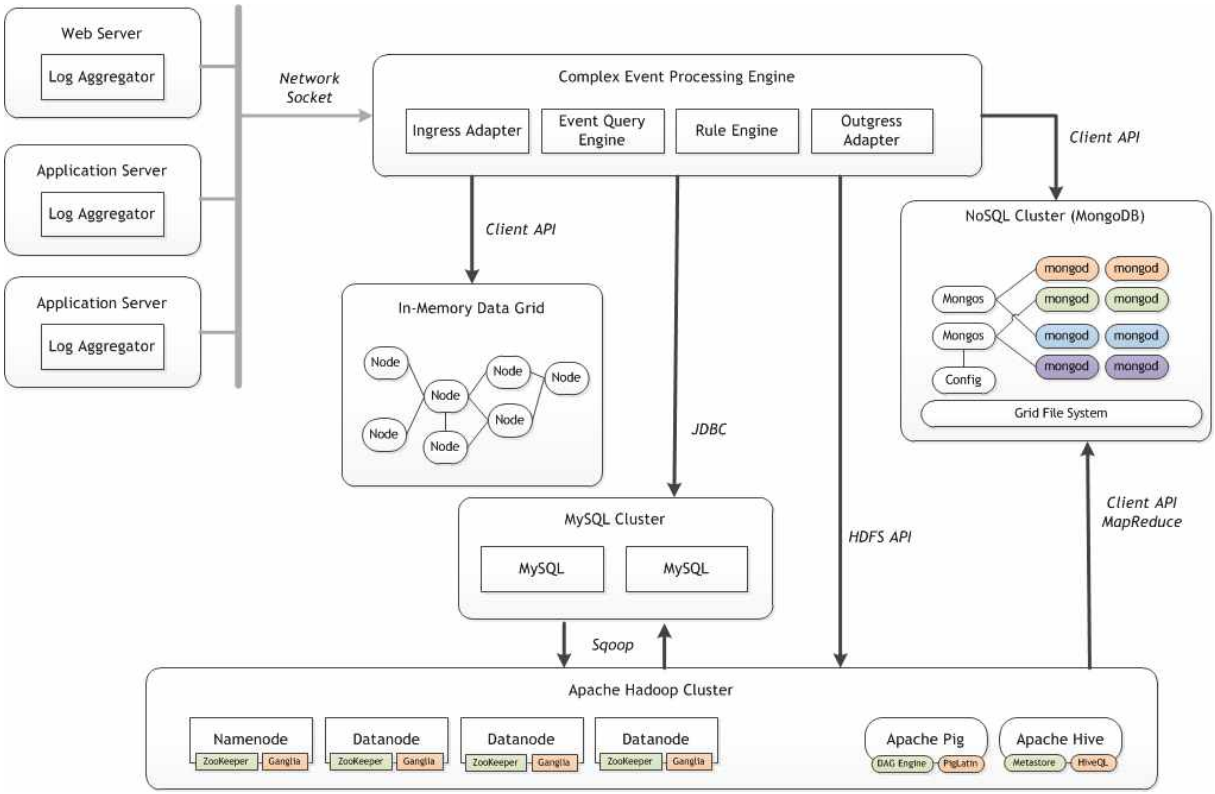
1. Complex Event Processing 아키텍처

리얼타임 스트리밍 기반 아키텍처에서 중요한 역할을 하는 기술인 CEP(Complex Event Processing)는 전통적으로 제조 및 금융 등에 활용해 왔는데 그 중에서 오픈소스로 크게 두각을 보이며 시장에 알려진 것은 Codehaus에 공개되어 있는 Esper와 JBoss Community에 공개되어 있는 Drools Fusion이 있다. 이 두 오픈소스는 서로 다른 기능을 제공하지만 시장에서 오픈소스로서 상당한 장악력이 있는 오픈소스이다.

가. Esper CEP

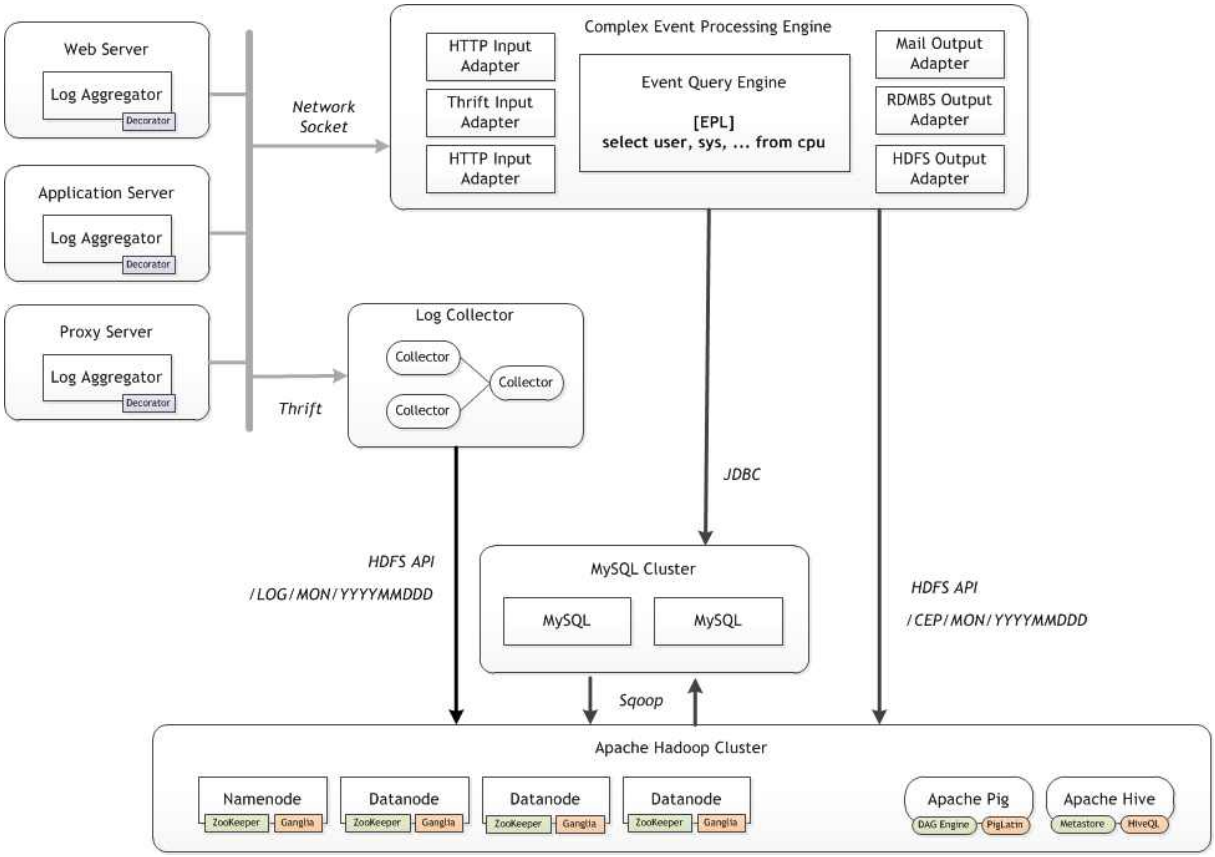
1) Esper CEP 기반 참조 아키텍처

Esper CEP는 세계적으로 많이 알려져 있는 CEP 엔진으로 GPL 라이센스를 가진 오픈소스이다. 주요 특징으로 높은 성숙도, 고성능, SQL like query engine, 단순함을 둘 수 있다. CEP 엔진이 scale up 특성을 가지고 있어서 종단점에 대량의 클라이언트가 직접 이벤트를 발생시키는 경우 시스템의 용량이 매우 커져야 한다. 다음은 CEP를 기반으로 한 시스템 아키텍처이다.



클라이언트의 Log Aggregator가 network socket을 이용하여 CEP 엔진에 이벤트를 발생시키면 Esper CEP 엔진은 ingress adapter를 통해 이벤트를 수신하고 이 이벤트는 Event Query Engine을 이용하여 특정한 이벤트만 추출하게 된다. 이후 요구사항에 따라서 NoSQL, RDBMS, In-Memory Grid로 데이터를 적제하거나 결합할 수 있으며 필요하다면 Hadoop의 HDFS에 로그를 취합할 수 있다.

다음은 실제 이런 아키텍처를 바탕으로 실제 현장에서 시스템의 리소스를 수집하여 실시간으로 수집하여 처리하는 시스템의 아키텍처이다.



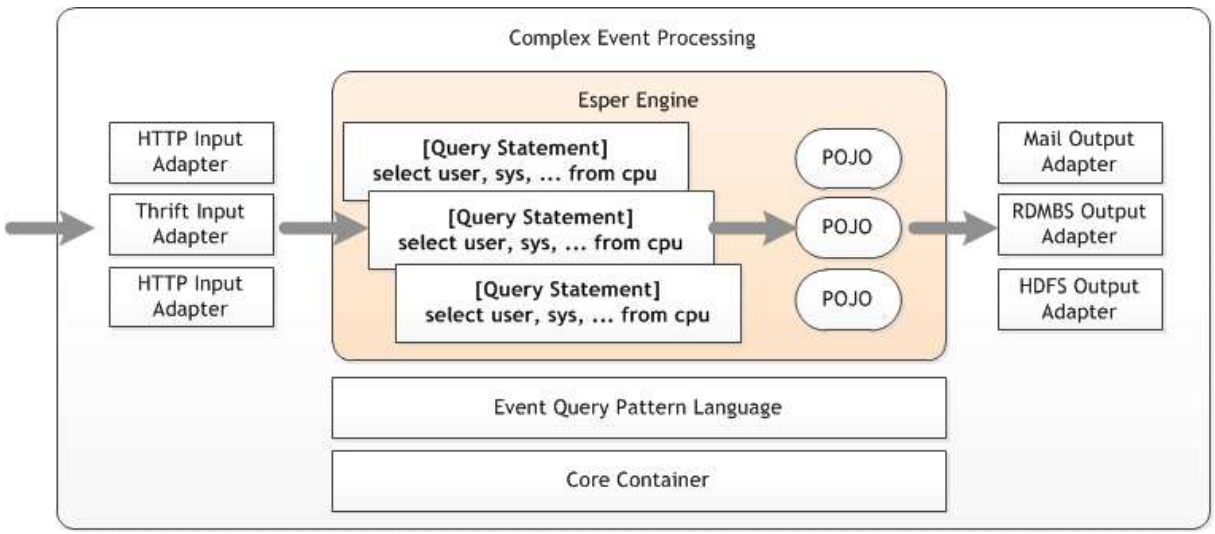
시스템의 각 구성요소는 다음의 역할을 수행한다.

구성요소	역 할
Log Aggregator	로그를 생성하는 서버에 설치하는 프로세스로 로그를 로딩하여 CEP 및 Log Collector로 송신하는 역할을 한다.
Input Adapter (Ingress | Ingest)	CEP 엔진의 구성 요소로서 Log Aggregator가 송신한 로그 메시지를 수신하는 역할을 한다. 수신한 로그 메시지는 Event Query Engine으로 전달하여 이벤트를 처리할 수 있도록 한다.
Log Collector	Log Aggregator가 송신한 로그를 분산 로그 수집기를 통해 수집하여 HDFS에 적재하는 기능을 수행한다.
Decorator	Log Aggregator의 일부로 동작하며 비정형 데이터를 정형 데이터로 변환하거나 로그 데이터를 수정하는 등의 작업을 수행한다.
MySQL Cluster	CEP 엔진 및 Hadoop과 연계하여 동작하며 CEP 엔진의 메타 데이터 및 수집한 로그 데이터를 저장하는 기능을 수행한다.

Apache Hadoop	Log Collector 및 CEP Engine에서 적재한 로그를 분석하고 저장하는 역할을 수행한다.
Output Adapter	CEP 엔진에서 EPL 조건에 따라서 처리한 이벤트를 외부로 전달하는 역할을 수행한다.
Event Query Engine	Input Adapter에서 전달한 이벤트를 처리하는 핵심 엔진이다.
EPL	SQL like 형식의 이벤트 쿼리로서 이 쿼리 조건에 따라서 input adapter로 들어온 이벤트를 실시간 수집한다.

2) Esper CEP의 구현 개념

Esper CEP는 내부 핵심 코어가 상당히 단순하게 구성되어 있어서 이해하기 매우 쉬운 오픈소스이면서 실제로 현장에 도입할 때 학습곡선이 상당히 낮은 오픈소스 중 하나이다. Esper CEP Engine의 내부 구조는 다음과 같다.



가) 이벤트 정의

Esper CEP에서는 이벤트를 자바의 POJO(Plain Old Java Object)로 정의한다. 상품의 가격 정보가 실시간으로 수집되는 경우 다음과 같이 정의한다.

```
package org.myapp.event;

public class OrderEvent {
    private String itemName;
    private double price;

    public OrderEvent(String itemName, double price) {
        this.itemName = itemName;
        this.price = price;
    }

    public String getItemName() {
        return itemName;
    }

    public double getPrice() {
        return price;
    }
}
```

나) 이벤트 처리

이벤트 처리는 다음과 같이 UpdateListener를 구현하여 등록하면 되고 new와 old 이벤트가 배열로 전달된다.

```
public class MyListener implements UpdateListener {
    public void update(EventBean[] newEvents, EventBean[] oldEvents) {
        EventBean event = newEvents[0];
        System.out.println("avg=" + event.get("avg(price)"));
    }
}
```

다) EPL 정의

EPL과 이벤트 POJO를 정의하면 이 모든 것을 다음과 같이 Esper Engine에 등록한다. 이제 외부에서 OrderEvent가 발생하면 최근 30초 내에 모든 가격정보의 평균값을 이벤트 핸들러로 전달한다.

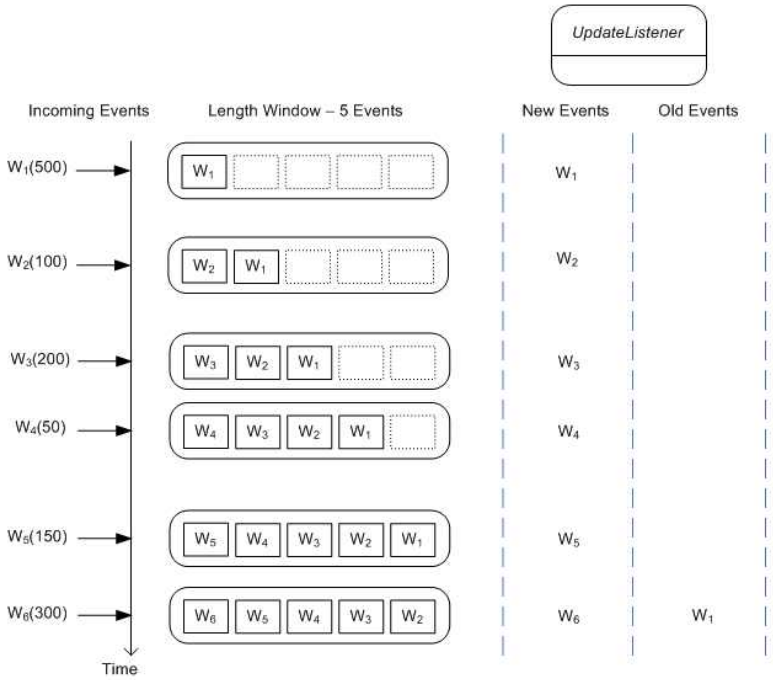
```
EPServiceProvider epService = EPServiceProviderManager.getDefaultProvider();
String expression = "select avg(price) from org.myapp.event.OrderEvent.win:time(30 sec)";
EPStatement statement = epService.getEPAdministrator().createEPL(expression);

MyListener listener = new MyListener();
statement.addListener(listener);
```

3) Length Window

CEP에서 중요한 개념이 length window이다. CEP는 특정한 길이 또는 시간만큼 이벤트 스트림을 유지하는 개념을 포함하고 있다. 예를 들어 계좌인출 이벤트를 5개만 유지하도록 하는 경우 다음과 같이 EPL을 작성할 수 있다.

```
select * from Withdrawal.win:length(5)
```

이 개념은 특정한 이벤트를 개수만큼 유지할 수 있는 장점이 있으며 이 개수를 넘어서게 되면 과거 이벤트는 소멸된다. 예를 들어 length window는 총 5개로 유지되도록 설정되어 있는 상황에서 다음과 같이 이벤트가 순차적으로 W1, W2, W3, W4, W5가 생성된다. 총 5개의 이벤트만 처리할 수 있으므로 W6가 되면 W1은 소멸된다. 이러한 처리 방식은 평균주가를 계산하거나 서비스의 처리량 등을 계산하는데 유용하다.



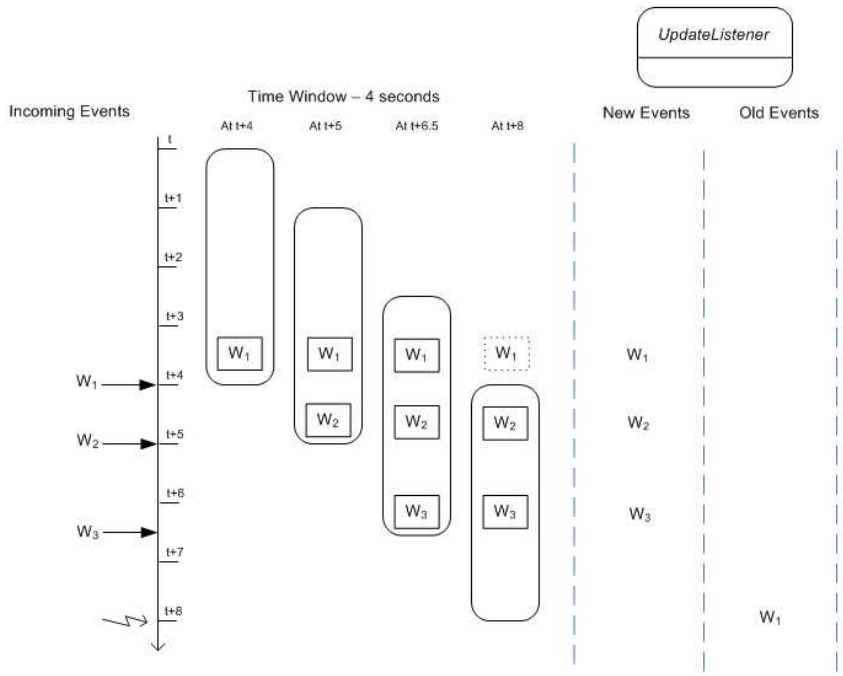
4) Time Window

CEP에서 length window 만큼 중요한 개념이 time window이다. 이 window는 지정한 시간만큼 이벤트 스트림을 유지한다. 예를 들면 최근 4초 내에 특정 계좌에서 인출한 평균값을 계산하는 경우 다음과 같이 EPL을 작성할 수 있다.

```
select account, avg(amount)
from Withdrawal.win:time(4 sec)
group by account
having amount > 1000
```

이 개념은 특정한 이벤트를 지정한 시간만큼 유지할 수 있는 장점이 있으며 이 개수를 넘어서게 되면 과거 이벤트는 자동으로 소멸된다. 예를 들어 다음과 같이 이벤트가 1초에 한 번씩 순차적으로 W1, W2, 건너뜀, W3, 건너뜀, W4가 생성된다. Time window가 4초간 유지되므로 4초가 지나면 제일 먼저 생성된 W1 이벤트가 소멸된다. 이렇게 이벤트가 4초간 지나면 4초 이내에 현재 있는 이벤트는 W2와 W3이다. 이런 방식으로 시간의 흐름에 따라서 이벤트가 일정한 시간동안 유지하고 이 이벤트 내에서 적절하게 selection, aggregation을 수행

할 수 있다. 특정한 비즈니스 요구사항에 있어서 이것만큼 좋은 대안은 없다. 다만 가장 큰 문제는 scale up 방식의 확장이다. 그래서 CEP를 사용하려면 현재로서는 좋은 사양과 성능을 제공하는 장비(CPU, Memory)를 구매하는 것이 가장 좋다.



4) Esper CEP 도입시 고려사항

항목	고려사항
성능	- Scale up 기반 아키텍처를 수용하므로 제한된 장비에서 성능을 최대한 발휘하기 위해서 복잡한 이벤트 처리를 지양할 것 - Input Adapter는 되도록 고성능 프로토콜을 사용할 것(권장: TCP/IP) - RDBMS와 EPL을 연계하거나 기타 외부 리소스와 연계할 대 상대적으로 매우 느린 성능의 외부 리소스는 지양할 것
확장성	- Scale up 기반 확장을 수행하므로 Scale out 형태의 확장을 하고자 하는 경우 상용 버전 사용을 권장함

	- Scale out 효과를 내려면 되도록 이벤트의 유형이나 특성에 따라서 Engine을 분리하여 처리할 것을 권장함
이중화	- 오픈소스 버전은 이중화를 구현할 수 없음 - 상용 버전 사용을 권장함
외부 데이터소스와 연계	- 처리한 이벤트를 외부 데이터 소스와 연계하여 처리하는 경우 비동기 방식을 사용할 것
온라인 서비스 변경	- 온라인 서비스 중 EPL을 변경하거나 하는 경우 OSGi 번들링 형태의 패키지 방식의 사용을 권장함

나. Drools Fusion CEP

JBoss Community의 LGPL 라이센스로 제공하는 Drools Fusion CEP는 사실 오픈소스 BPM인 jBPM의 구성 요소이다. 이 오픈소스는 Esper와는 다르게 보험사 등에서 사용하는 rule을 기반으로 동작한다. Esper와 차별성이라면 복잡한 rule을 지원하는 것이다. 고성능 처리를 목표로 하지는 않지만 복잡한 규칙을 실시간 이벤트 처리하는데 있어서 최적의 대안이다.

2. 분산 스트리밍 아키텍처

가. STORM

- 간편한 프로그래밍 모델. MapReduce가 병렬 배치 처리 수행의 복잡도를 낮춰주는 것과 같이 실시간 분산 처리 수행의 복잡도를 낮춘다.
- 어떤 프로그래밍 언어이든 활용 가능. Storm 상에서 프로그래밍 언어의 제한없이 사용가능하다. 기본적으로 클로저, 자바, 루비, 파이썬 언어를 지원한다. 그 밖에 다른 언어는 간단한 Storm 통신 프로토콜 구현으로 추가 가능하다.
- 장애에 안전. Storm은 워커 프로세스와 노드의 실패를 자동으로 관리한다.
- 수평적 확장 용이. 멀티 스레드, 프로세스, 서버를 사용하여 병렬 처리 계산을 수행한다.
- 메시지 처리 보장. Storm은 각 메시지가 손실 없이 전체 프로세스에서 완전하게 최소 한 번 이상의 완전한 처리를 보장한다. 작업 실패 시 데이터 소스에서부터 다시 메시지 재처리가 가능하다.
- 빠르다. 메시지가 빠르게 처리될 수 있도록 설계되었다. 내부적으로 ZeroMQ를 메시지 큐로 사용한다.
- 로컬 모드. Storm은 "로컬 모드"를 제공한다. "로컬 모드"는 Storm 클러스터 기반의 실행을 완전하게 시뮬레이션 한다. 이 기능은 Storm 토폴로지의 개발과 단위 테스트를 용이하게 한다.

나. Apache S4

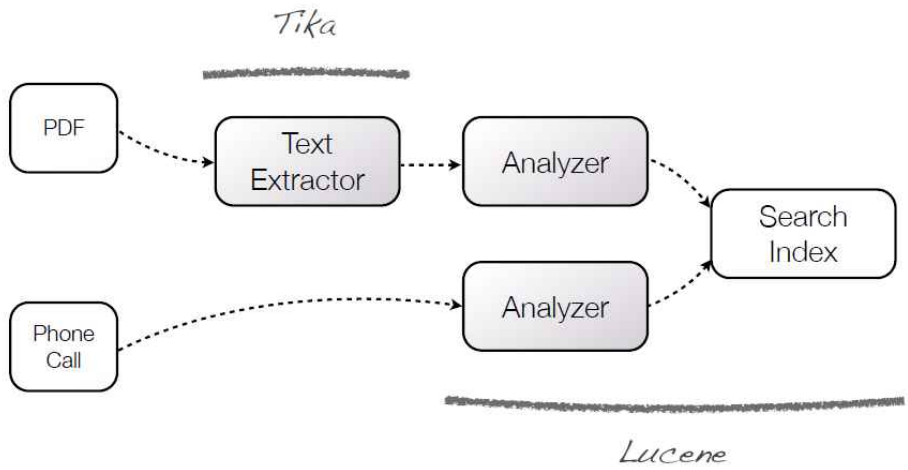
Apache S4는 Yahoo!에서 개발하여 Apache License 하에서 배포하고 있는 오픈소스 프로젝트이다. 이 프로젝트는 Yahoo!에서 실시간 검색 엔진을 구현하는 요구사항으로 개발된 오픈소스로서 경량이면서 단순한 아키텍처로 구성되어 있다. 간단한 집계 기능을 수행하고 이벤트 스트림을 가공하는 작업을 수행하기 때문에 이벤트 스트림의 연산 작업이 주요 기능적 요건인 경우 적용하기 적합한 오픈소스이다.

- Yahoo! Labs에서 2008년에 시작하여 2009년 오픈소스로 공개하였다. 2011년 11월에 Apache Incubator에 추가되어 현재 개발 중이다.
- Real-Time으로 검색 광고를 개인화하기 위한 needs가 있어서 개발
- 적용 분야
  - O 개인화 검색
  - O 트위터 트렌드
  - O 스팸 필터링
  - O 센서 네트워크
  - O 네트워크 침임 감지
  - O 시장 예측 및 자동 트레이딩

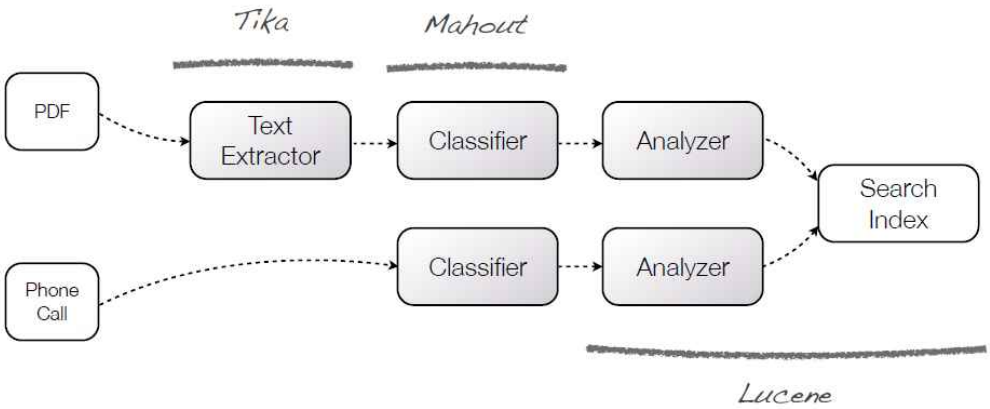
다음은 검색 엔진의 기본 화면으로 일반적으로 검색 엔진은 다음과 같이 검색 인덱스를 생성하는 주기가 상당히 길다. 물론 검색을 위해서 PageRank 알고리즘을 적용하는 경우 각 문서 간 그래프 연산을 수행해야 하므로 실시간 검색을 할 수 없는 경우도 있지만 실시간 검색을 적용하여 이 시간을 줄이는 것은 서비스 품질에 있어서 매우 중요하다.

이벤트 스트림(파일 또는 전화통화 등등)이 발생하면 일반적으로 검색을 위해서 검색 엔진

에 인덱스를 생성해야 하나 이 작업은 배치 작업을 통해서 수행하는 것이 일반적이다. 이러한 인덱스 생성 작업은 다음과 같이 하나 이상의 단계를 거쳐야 한다. 문제는 각 단계가 CPU 리소스를 많이 점유하는 작업들로 구성되어 있어서 확장에 있어서 문제가 발생하고 통상적으로 인덱싱 작업은 배치 작업 형태로 동작하기 때문에 실시간 검색에 적용하는데 근본적인 문제가 있다는 것이다.



또한 보다 정교한 검색을 위해서 다음과 같이 중간에 분류 작업을 추가하는 경우 배치 작업의 확장성 또한 문제가 발생하고 더욱더 많은 시간이 소요된다.



이러한 문제를 극복하기 위해서 검색 인덱스를 생성하기 위한 각 기능을 하나 이상의 노드로 분산시키고 동시에 동일한 기능을 수행노드를 중복하여 구성함으로써 노드가 장애 시 대응할 수 있도록 하는 아키텍처를 구현하였다.

1) 주요 특징

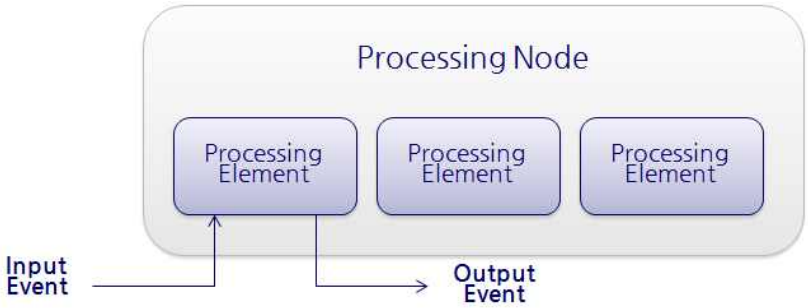
Apache S4 도입 시 가장 중요한 요구사항은 바로 확장성과 장애 대응 능력이다. Apache S4는 Apache ZooKeeper를 이용하여 Cluster Management를 수행함으로써 동작중인 노드에서 장애가 발생하는 경우 자동으로 fail-over를 수행하고 노드가 추가되거나 삭제되는 경우 자동으로 부하를 분산시키는 능력을 가지고 있다.

Scalable 노드 추가 시 선형적 성능 향상	**Cluster Management** ZooKeeper를 이용한 클러스터 관리	**Extensible** 단순한 API를 이용하여 애플리케이션을 개발하고 배포
Decentralized No Single Point Failure	**Partial Fault-Tolerance** 장비 장애 시 자동으로 Stand-By Server 활성화	**Proven** Yahoo! 검색 운영 시스템에 적용

기본적으로 분산 형태로 동작하므로 동작중인 노드에서 장애가 발생하는 경우 특정 노드의 장애가 전체 시스템으로 확산하는 것을 방지하며 단순한 형태의 API를 제공하여 개발이 용이한 장점을 가지고 있다.

2) PE(Processing Element)

PE(Processing Element)는 Apache S4에서 핵심 기능으로 실시간으로 들어오는 이벤트 스트림을 처리하는 자바 기반 POJO(Plain Object Java Object)이다. 이 PE의 역할은 이벤트를 가공하는 역할을 수행하며 각 노드에는 하나 이상의 PE를 통해 이벤트 스트림을 순차적으로 처리할 수 있다.

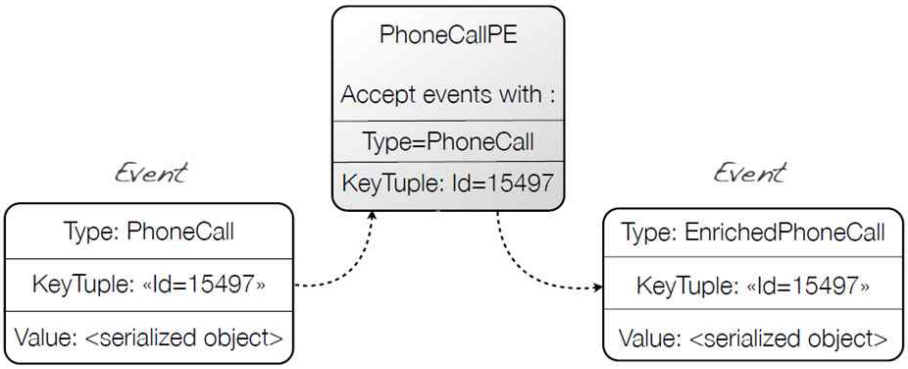


PE(Processing Element)는 기반으로 AbstractPE 클래스를 확장하여 개발하며 이 클래스에는 processEvent(Object object) 콜백 메소드가 있으며 이 메소드는 이벤트 스트림을 수신했을 때 Apache S4에서 호출한다. 개발자는 이 메소드에서 적절하게 다른 이벤트로 변경하거나 이벤트를 처리하면 된다.

```
public class SentenceReceiverPE extends AbstractPE {

  public void processEvent(Sentence sentence) { // 이벤트 처리
      System.out.printf("Sentence is '%s', location %s\n", sentence.getText(),
                        sentence.getLocation());
  }

  @Override
  public void output() {
      // not called in this example
  }

  @Override
  public String getId() {
      return this.getClass().getName();
  }
}
```

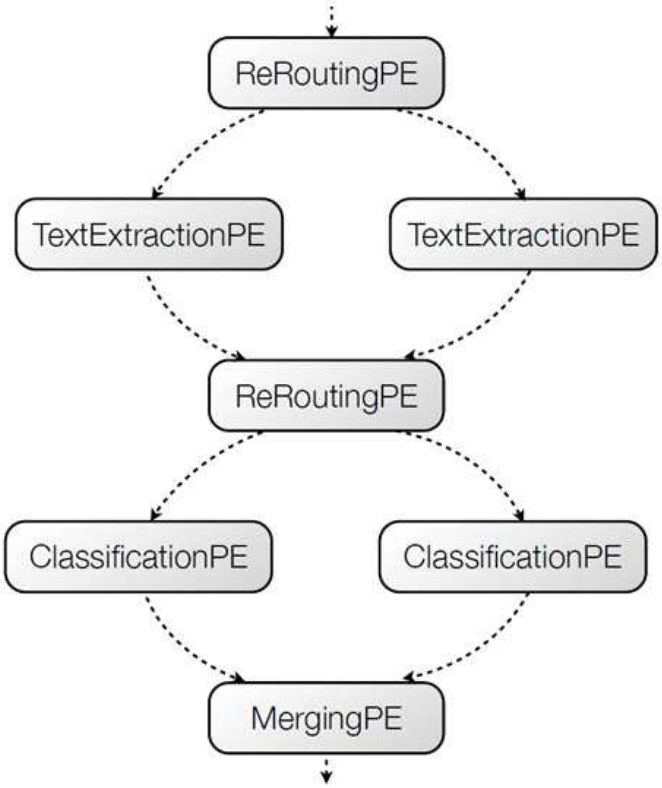
2) 이벤트 처리

PE(Processing Element)는 이벤트를 처리하는 이벤트 핸들러로써 다음과 같이 입출력 이벤트를 갖는다. 다음의 경우 입력 이벤트인 PhoneCall은 PhoneCallPE를 통해 처리한 이후에는 다음과 같이 새로운 PhoneCall 이벤트인 EnrichedPhoneCall 이벤트가 생성된다. 이렇게 하나 이상의 PE(Processing Element)를 구현하여 이벤트를 순차적으로 처리할 수 있도록 구현할 수 있다.



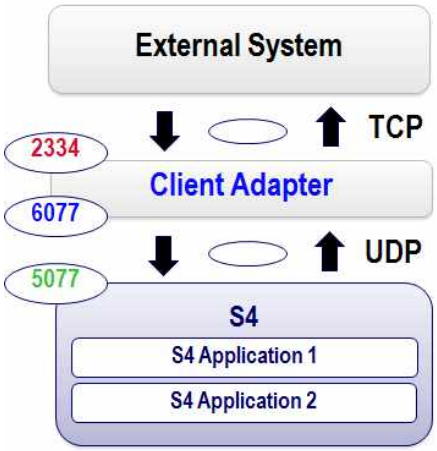
3) PE Pipeline

이벤트를 처리하는 PE(Processing Element)는 개발자가 직접 개발하지만 분산 환경에서 이벤트 스트림을 다수의 노드에 분산 시키는 PE(Processing Element)는 Apache S4에서 제공한다. 다음의 경우 ReRoutingPE는 이벤트를 조건에 따라서 다수의 노드로 분산 시키는 작업을 하며 이 작업은 scale out을 구현하는 핵심 요소가 된다. 이벤트는 ReRoutingPE를 통해 개발자가 작성한 PE(Processing Element)인 TextExtractPE로 전달되고 다시 ReRoutingPE를 통해 ClassificationPE로 전달된다. 그리고 최종적으로 MergingPE로 이벤트가 취합된다. 이런 형식으로 이벤트는 다수의 노드에 분산되어 PE(Processing Element) 파이프라인을 구성한다.



4) Client Adapter

PE(Processing Element)가 이벤트를 처리하는 핵심 구성 요소이지만 이것의 시작점은 바로 이벤트를 수신하는 Client Adapter이다. Client Adapter는 외부 시스템과 통신을 통해서 이벤트 스트림을 수신하여 내부 Apache S4 애플리케이션으로 전달된다(Apache S4 애플리케이션은 PE(Processing Element)가 배포되어 있는 시스템이다). 여기에서 중요한 것은 고성능 네트워킹을 수행하기 위해서 Client Adapter와 외부 시스템은 TCP를 사용하고, Client Adapter와 Apache S4 애플리케이션은 UDP를 사용한다는 점이다. UDP는 TCP에 비해서 상대적으로 네트워크 비용이 많이 들지만 고성능 네트워킹을 구현할 수 있으므로 대부분의 클러스터링 기반 인프라 시스템에서 사용하는 기법이다. Apache S4도 Client Adapter를 통해서 전달받은 이벤트 스트림을 내부 Apache S4 애플리케이션에 전달하기 위해서 UDP를 사용한다.



s4-core/conf/default/client-stub-conf.xml

```
<property name="connectorPort" value="2334"/>
```

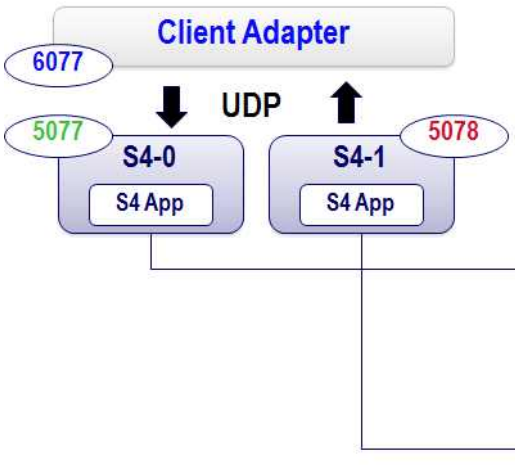
s4-core/conf/default/clusters.xml

```
<config version="-1">
  <cluster name="s4" type="s4" mode="unicast">
    <node>
      <partition>0</partition>
      <machine>localhost</machine>
      <port>5077</port>
      <taskId>s4node-0</taskId>
    </node>
  </cluster>
  <cluster name="client-adapter" type="s4" mode="unicast">
    <node>
      <partition>0</partition>
      <machine>localhost</machine>
      <taskId>client-adapter-0</taskId>
      <port>6077</port>
    </node>
  </cluster>
</config>
```

상기 도식을 보면 Client Adapter와 Apache S4 애플리케이션이 배포되어 있는 S4 인스턴스(s4node-0)는 파티션 0로 구성되어 있으며 이와 동일한 파티션에 Client Adapter(client-adapter-0)가 연결되어 있다. 이로써 Client Adapter와 S4 노드는 UDP를 기반으로 메시지를 송수신할 수 있게 된다.

5) Multi-node Clustering

다수의 노드에 S4 애플리케이션을 배포할 수 있지만 동일한 노드에 서로 다른 포트로 S4 애플리케이션이 배포되어 있는 S4 애플리케이션의 인스턴스를 하나 이상 동작시켜 부하를 분산시킬 수 있다. 최근 판매하는 대부분의 장비가 6 Core × 2 CPU를 가진 서버 장비가 많으므로 하나의 노드에 하나의 S4 애플리케이션 인스턴스를 동작시키는 것보다는 시스템의 용량을 최대한 사용하기 위해서 다수의 S4 애플리케이션 인스턴스를 동작시키는 것이 바람직하다(특히 S4는 Disk I/O를 발생시키지 않고 주로 In-Memory 기반 동작을 수행하므로 많은 양의 메모리를 구성하는 것이 더 적합하다).



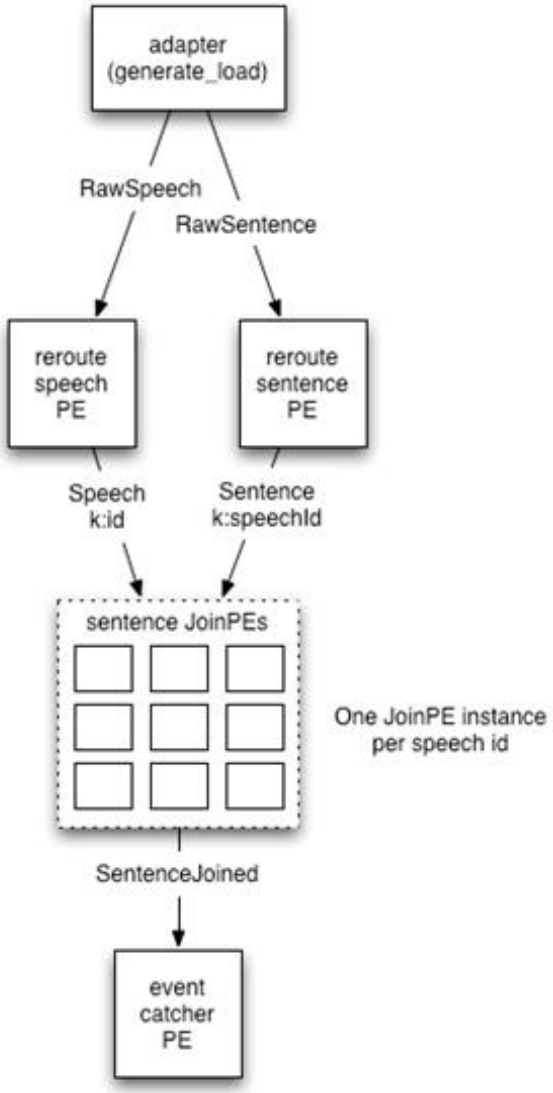
s4-core/conf/default/clusters.xml

```
<config version="-1">
  <cluster name="s4" type="s4" mode="unicast">
    <node>
      <partition>0</partition>
      <machine>localhost</machine>
      <port>5077</port>
      <taskId>s4node-0</taskId>
    </node>
    <node>
      <partition>1</partition>
      <machine>localhost</machine>
      <port>5078</port>
      <taskId>s4node-1</taskId>
    </node>
  </cluster>
</config>
```

상기 도식을 보면 하나의 노드에 서로 다른 포트를 가진 S4 애플리케이션 인스턴스가 두 개 존재하는 것을 확인할 수 있으며 서로 다른 파티션을 가진 것으로 확인할 수 있다. 이 전략은 서로 다른 S4 애플리케이션을 하나의 장비에 다수 배포하는 전략으로써 시스템의 리소스를 최대한 사용하고 성능을 높이기 위함이다.

6) Reroute & Merge

앞서 설명한 것처럼 reroute는 메시지를 유형별로 분배하는 기능으로써 S4 자체에서 제공하는 PE이다. Merge 또한 PE의 처리 결과를 하나로 결합하는 S4 자체 제공 PE이다. 이 두 기능을 활용하면 부하를 분산시키고 그 결과를 취합하는 Hadoop의 MapReduce 기법을 어느 정도 구현할 수 있다. 다음의 경우 Client Adapter를 통해 수신한 Speech와 Sentence 이벤트 스트림을 서로 다른 ReRoutePE로 전달하여 다수의 노드에서 분산하여 처리하도록 하였고 그 처리 결과를 Merge를 통해서 하나의 이벤트로 결합하였다.



분산 환경에서 동작하는 이 구성 기법은 다음의 SQL Query를 S4로 구현한 것이다.

```
select Sentence.*, Speech.location
into SentenceJoined
from Sentence, Speech
where Sentence.speechId = Speech.id
```

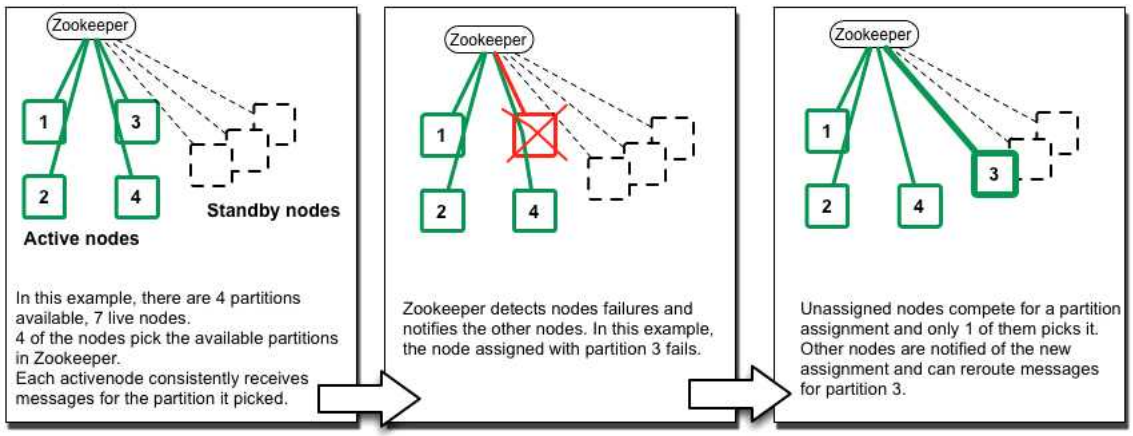
7) Fault Tolerance

이벤트 스트리밍 처리는 일반적으로 장시간 동작하는 애플리케이션으로써 시간이 지나면서 데이터가 누적되거나 다수의 노드가 중지되거나 재시작되는 경우가 흔하게 발생한다. 최악의 경우 모든 노드가 shutdown되어 시스템이 부분적으로 또는 전체가 동작하지 못할 수 있으며 이 경우 메모리의 이벤트 상태를 손실할 수 있다. Apache S4에서는 이런 문제를 해결하기 위해서 다음의 기능을 제공한다.

- 고가용성
- 체크포인트 기반 상태 복구
- 상태를 보존하는 동안 최소 처리 지연

가) 고가용성

Apache S4는 갑자기 특정 노드에서 문제가 발생한 경우 노드의 문제를 감지하고 자동으로 메시지를 대기중인 타 노드로 redirect 처리한다. 이것을 도식으로 표현하면 다음과 같다.



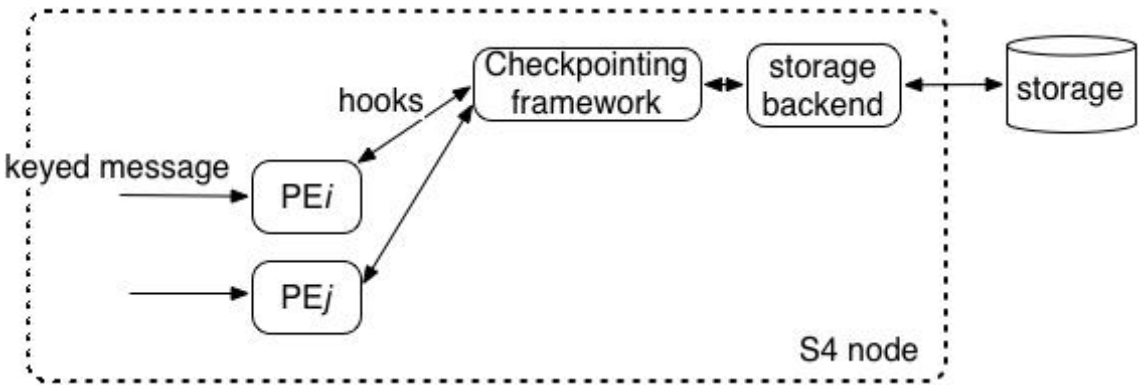
우선 총 4개의 파티션에 7개의 노드가 현재 동작중인 상태에서 3번 노드의 이상을 ZooKeeper가 감지한 경우 자동으로 타 노드에 3번 노드로 부여하고 동작시킨다. 이것은 시스템의 가용성을 높이기 위한 매우 중요한 작업으로 이 모든 과정에 ZooKeeper의 상태 관리 및 통지 기능이 관여한다. 따라서 분산 환경에서 노드의 동작 상태 모니터링 및 에러 탐지 등의 작업을 통해 고가용성을 구현하려면 반드시 ZooKeeper를 이용해야 한다.

나) 체크포인트 기반 상태 복구

장애가 발생하여 이벤트 스트림을 타 노드로 전달하는 경우 해당 노드가 이전 상태를 기억하지 못한다면 이벤트 스트림을 처리하는데 있어서 상태의 일관성이 깨지게 되므로 큰 문제가 된다. Apache S4는 이런 상태의 일관성 문제를 해결하기 위해서 이전 상태를 복원하는데 다음의 두 가지 방법을 사용한다.

- 주기적으로 PE의 상태에 대한 체크포인트를 생성한다.
- 늦은 복구(메시지를 통해 트리거)

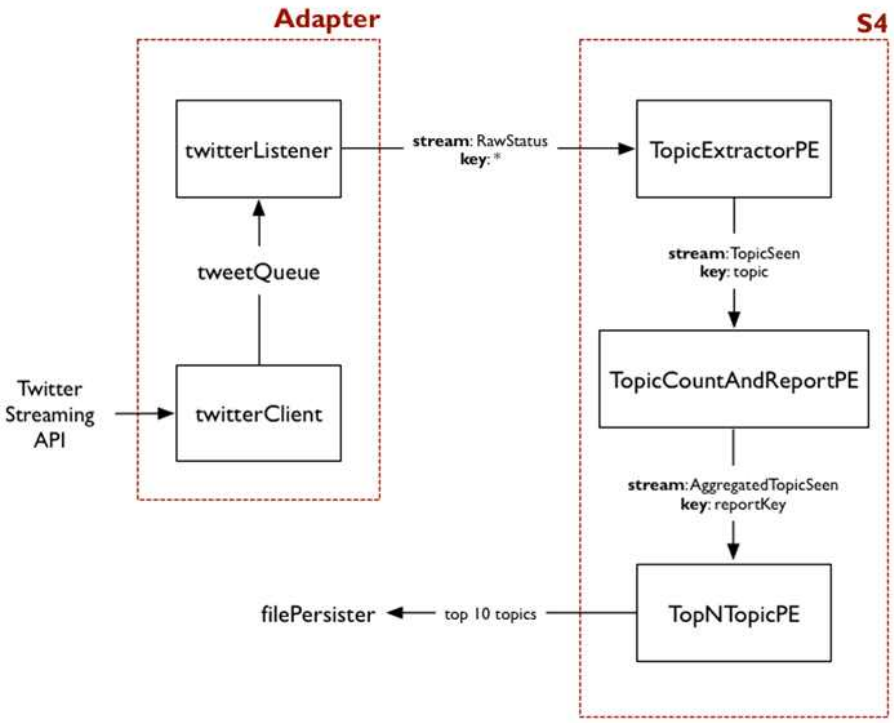
지연을 최소화하기 위해서 우선 체크포인트를 생성하는 작업은 비동기로 수행하며 ZooKeeper 등의 Coordinator를 통해서 통제받지 않고 단독으로 수행한다. 그렇게 하여 PE 인스턴스를 직렬화하고 직렬화한 PE를 리모트 스토리지에 저장하는 과정을 수행한다. 복구는 리소스 사용을 최적화하기 위해서 복구한 S4 노드에 메시지가 도착했을 때 PE의 복구 작업을 진행한다.



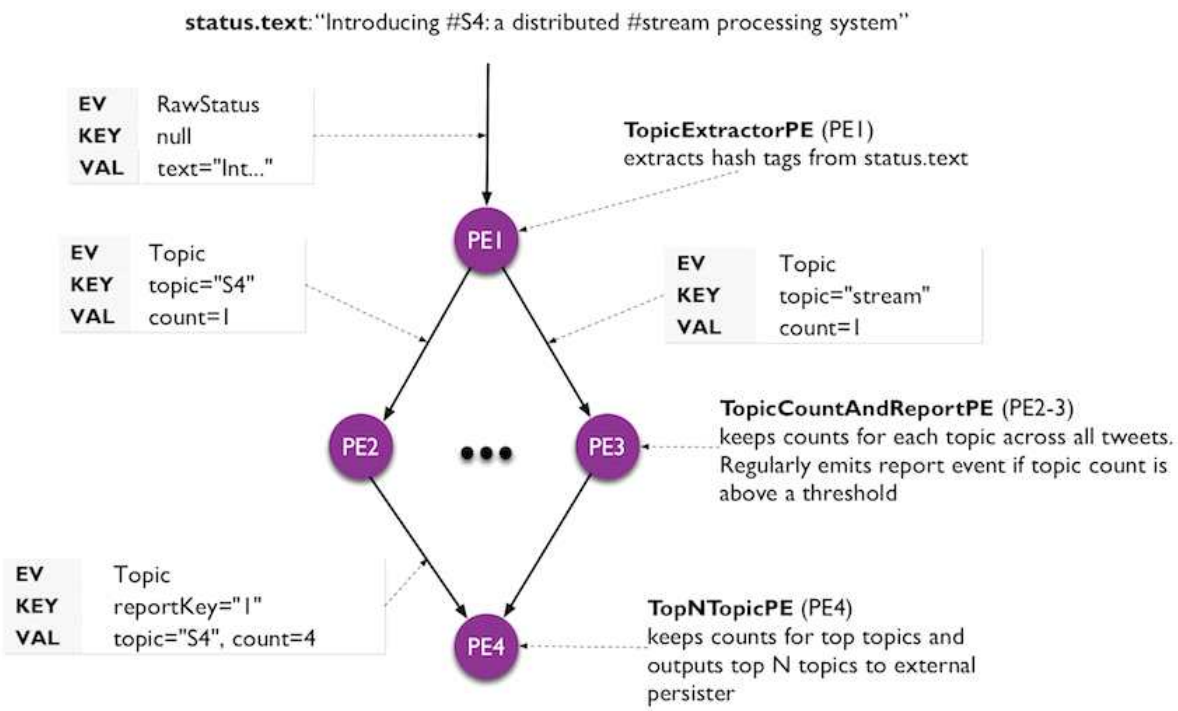
8) 사례 연구 (Twitter Topic Counter)

간단한 사례 연구를 통해서 Apache S4의 실시간 스트리밍 처리가 어떻게 동작하는지 살펴보자. 우선 Twitter에서 가장 인기 있는 토픽을 찾기 위해서는 Hadoop의 wordcount처럼 토픽을 추출하여 해당 토픽별로 개수를 합산해야 한다. 그리고 나서 상위 Top N을 추출하면 된다. 이러한 작업을 위해서는 Client Adapter는 Twitter에서 제공하는 API를 통해 Twitter의

트윗을 스트리밍 해야 한다. 다음의 경우 Client Adapter는 Twitter Client와 TCP/IP로 통신하며 Twitter Client는 Twitter에서 제공하는 API를 이용하여 트윗을 스트리밍한다. 일단 트윗 이벤트가 발생하면 Client Adapter를 통해서 TopicExtractorPE로 트윗 이벤트가 들어가고 이때 이 PE는 토픽을 추출하여 TopicCountAndReportPE로 전달한다. 이 TopicCountAndReportPE는 각 토픽에 대해서 개수를 알아내고 TopNTopicPE로 이벤트를 전달하면 이 TopNTopicPE는 모든 토픽 중에서 가장 개수가 많은 상위 10개를 추출한다.



이러한 처리과정은 PE 관점에서 다음과 같이 다시 표현할 수 있다. TopicCountAndReportPE는 토픽의 개수를 계산해서 출력 이벤트를 생성하는 PE로서 각각의 토픽에 대해서 처리를 하므로 하나 이상 구성하여 분산 처리한다.

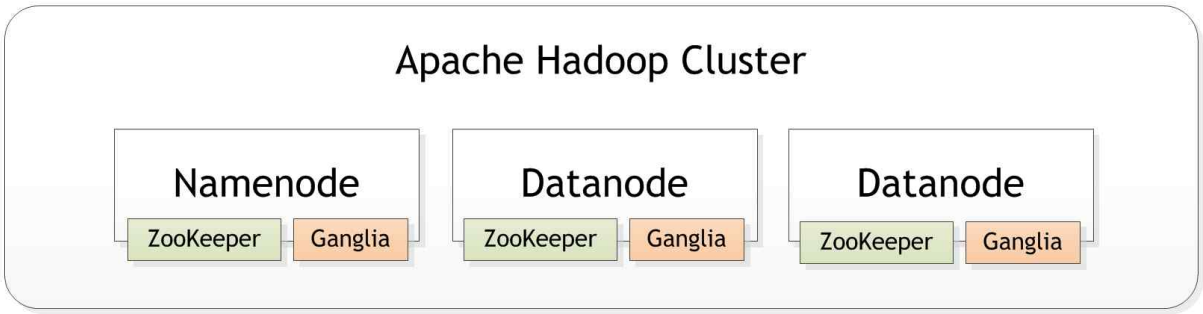

status.text: "Introducing #S4: a distributed #stream processing system"
EV RawStatus
KEY null
VAL text="Int..."
TopicExtractorPE (PE1)
extracts hash tags from status.text
PE1
EV Topic
KEY topic="S4"
VAL count=1
EV Topic
KEY topic="stream"
VAL count=1
PE2
PE3
TopicCountAndReportPE (PE2-3)
keeps counts for each topic across all tweets.
Regularly emits report event if topic count is
above a threshold
EV Topic
KEY reportKey="1"
VAL topic="S4", count=4
PE4
TopNTopicPE (PE4)
keeps counts for top topics and
outputs top N topics to external
persister

Appendix :: 제1절 Apache Hadoop 설치 가이드

본 문서는 중소기업과 같이 인력을 보유하지 않은 조직에서 Apache Hadoop을 도입하기 위한 첫 번째 관문인 설치 작업을 실제 현장의 경험자의 경험과 노하우를 바탕으로 설치할 수 있도록 지침을 제공하는 것을 목적으로 한다.

1. 시스템 구성

본 가이드는 다음과 같이 3대의 리눅스 장비가 갖추어졌다는 가정 하에서 Apache Hadoop을 어떻게 설치하는지를 설명한다. 실제 환경에서는 기본적으로 5대 이상의 장비를 권장하지만 본 설치 가이드에는 3대를 기준으로 설명한다. 또한 본 문서에서는 각 장비를 hadoop1, hadoop2, hadoop3로 칭한다.



2. 호스트 파일 설정하기

DHCP 또는 DNS 서버가 별도로 구축되어 있지 않은 경우 각 장비가 다른 호스트를 인지하도록 하기 위해서 호스트 파일을 설정해야 한다.

※ 많은 수의 Hadoop 장비를 계획하거나 구성하고자 한다면 DHCP와 DNS 서버 구축을 권장한다. 그렇지 않다면 많은 수의 장비를 관리하는 일은 시스템 운영자에게 상당한 부담으로 작용한다.

이를 위해서 hadoop1 서버의 /etc/hosts 파일에 3개의 서버와 호스트명을 다음과 같이 추가한다.

```
root:hadoop1 #> vi /etc/hosts
# Do not remove the following line, or various       programs
# that require network functionality will fail.
127.0.0.1                   localhost.localdomain       localhost
::1                         localhost6.localdomain6  localhost6
172.27.135.97 hadoop1
172.27.11.82 hadoop2
172.27.82.221 hadoop3
```

이제 hadoop1 장비에서 다음과 같이 정상적으로 다른 모든 노드가 ping이 되는지 확인한다.

```
root:hadoop1 #> ping hadoop2
PING hadoop2 (172.27.11.82) 56(84) bytes of data.
64 bytes from hadoop2 (172.27.11.82): icmp_seq=1 ttl=64    time=0.171 ms

--- hadoop2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time    0ms
rtt min/avg/max/mdev = 0.171/0.171/0.171/0.000 ms
root:hadoop1 #> ping hadoop3
PING hadoop3 (172.27.82.221) 56(84) bytes of data.
64 bytes from hadoop3 (172.27.82.221): icmp_seq=1 ttl=64    time=0.158 ms
```

설정이 완료되면 /etc/hosts 파일을 hadoop2, hadoop3에 적용하기 위해서 다음의 커맨드를 실행한다. scp 커맨드는 원격의 장비에 파일을 복사하는 커맨드이다.

```
root:hadoop1 #> scp /etc/hosts root@hadoop2:/etc
The authenticity of host 'hadoop2 (172.27.11.82)' can't    be established.
RSA key fingerprint is df:86:40:ed:43:bf:76:7e:9e:61:96:60:68:90:2f:f4.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'hadoop2,172.27.11.82' (RSA)    to the list of known hosts.
root@hadoop2's password: ********
hosts                                                 100%   253         0.3KB/s    00:00

root:hadoop1 #> scp /etc/hosts root@hadoop3:/etc
The authenticity of host 'hadoop3 (172.27.82.221)' can't    be established.
RSA key fingerprint is df:86:40:ed:43:bf:76:7e:9e:61:96:60:68:90:2f:f4.
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added 'hadoop3,172.27.82.221' (RSA)  to the list of known hosts.
root@hadoop3's password: *********
hosts                                          100%  253       0.3KB/s   00:00
```

3. Hadoop을 동작시킬 계정 생성하기

이제 Hadoop을 실행할 계정을 하나 생성한다. hadoop 이라는 계정이다. Cloudera Hadoop 배포판에서는 CDH3부터 Job Tracker와 HDFS를 운영하는 계정을 별도로 구조 관리한다. 이 구조를 따라가려면 두 개의 계정을 생성하고 하나의 계정으로 운영하려면 다음과 같이 하나만 생성한다. **이 작업은 모든 노드에서 root 계정으로 수행해야 한다.**

```
root:hadoop1 #> adduser hadoop                              → hadoop 계정 생성
Adding user `hadoop' ...
Adding new group `hadoop' (1000) ...
Adding new user `hadoop' (1000) with group `hadoop' ...
Creating home directory `/home/hadoop' ...
Password: ******
root:hadoop1 #> mkdir -P /usr/local/java                    → JDK를 설치할 디렉터리 생성
root:hadoop1 #> chown -R hadoop:hadoop /usr/local/java      → hadoop 계정으로 권한 변경
```

4. 서버 인증서 교환하기

Apache Hadoop을 실행하게 되면 리모트 커맨드를 실행하게 되므로 서로 신뢰하는 서버로 만들어 두지 않으면 계속 패스워드를 묻게 된다. 이 과정을 없애기 위해서 각 서버의 인증서를 교환하는 작업을 해야 한다. 이를 위해서 다음과 같이 스크립트를 작성하고 이 스크립트를 ssh-key-exchange_centos.sh 파일로 기록한다.

```
#!/bin/sh

KEYS_DIR=~/hadoop-ssh-keys

HOST_LIST=`cat hosts.txt`

rm -rf "${KEYS_DIR}"

mkdir -p "${KEYS_DIR}"

cd "${KEYS_DIR}"

if [ ! -e /root/.ssh/id_dsa ]; then
ssh-keygen -t dsa -P '' -f /root/.ssh/id_dsa;
```

```
fi

for host in ${HOST_LIST}
do
ssh ${host} "if [ ! -e /root/.ssh/id_dsa ]; then \
    ssh-keygen -t dsa -P '' -f /root/.ssh/id_dsa; fi"
scp ${host}:/root/.ssh/id_dsa.pub ${host}.pub
done

cat * > authorized_keys
cat /root/.ssh/id_dsa.pub >> authorized_keys

cp authorized_keys /root/.ssh/authorized_keys
chown root:root /root/.ssh/authorized_keys

for host in ${HOST_LIST}
do
scp authorized_keys ${host}:/root/.ssh/authorized_keys
ssh ${host} "chown root:root /root/.ssh/authorized_keys"
done
```

위 스크립트는 CentOS용으로 작성된 것이며 Ubuntu를 이용하는 경우 스크립트의 내용을 다음과 같이 수정한다. 다음의 스크립트는 Apache Hadoop을 설치하고 동작시키는 계정이 hdfs 계정이라는 가정 하에서 작성되었다. 만약 계정이 다르다면 다음의 스크립트에서 /home/hdfs와 hdfs 계정명으로 모두 원하는 계정명으로 변경해야 한다.

```
#!/bin/sh

KEYS_DIR=~/hadoop-ssh-keys                    ①

HOST_LIST=`cat hosts.txt`                     ②

rm -rf "${KEYS_DIR}"

mkdir -p "${KEYS_DIR}"

cd "${KEYS_DIR}"

if [ ! -e /home/hadoop/.ssh/id_dsa ]; then
  sudo -u hadoop ssh-keygen -t dsa -P '' -f /home/hadoop/.ssh/id_dsa;          ③
fi
```

```
for host in ${HOST_LIST}
do
    ssh ${host} "if [ ! -e /home/hadoop/.ssh/id_dsa ]; then \
        sudo -u hadoop ssh-keygen -t dsa -P '' -f /home/hadoop/.ssh/id_dsa; fi"      ④
    scp ${host}:/home/hadoop/.ssh/id_dsa.pub ${host}.pub
done

cat * > authorized_keys
cat /home/hadoop/.ssh/id_dsa.pub >> authorized_keys                                  ⑤

cp authorized_keys /home/hadoop/.ssh/authorized_keys
chown hadoop:hadoop /home/hadoop/.ssh/authorized_keys

for host in ${HOST_LIST}
do
    scp authorized_keys ${host}:/home/hadoop/.ssh/authorized_keys                    ⑥
    ssh ${host} "chown hadoop:hadoop /home/hadoop/.ssh/authorized_keys"              ⑦
done
```

상기 스크립트의 각 부분은 다음의 동작을 수행한다.

① 각 서버의 SSH Key를 저장할 디렉터리

② 각 서버의 IP 주소를 담고 있는 텍스트 파일

③ 이 스크립트를 실행하는 서버의 hdfs 계정에 대한 SSH Key를 생성

④ 각 서버의 hdfs 계정에 대한 SSH Key를 생성하고 각 서버의 공개키를 현재 서버로 가져옴

⑤ 모든 서버의 공개키를 authorized_keys 파일에 저장

⑥ authorized_keys 파일을 각 서버로 전송

⑦ 파일을 hdfs 계정이 있을 수 있도록 권한을 수정

※ SSH Key 교환 스크립트는 반드시 ROOT 계정으로 실행하도록 하고 모든 서버는 동일한 디렉터리에 동일한 구성을 권장한다.

hosts.txt 파일에 Hadoop을 설치할 서버의 내부 IP를 hadoop1을 제외하고 다음과 같이 추가

한다.

```
root:hadoop1 #> vi hosts.txt
...
root:hadoop1 #> cat hosts.txt            → vi와 같은 편집기로 파일에 IP 주소를 추가
172.27.11.82
172.27.82.221

root:hadoop1 #> ll
total 44
-rw-------- 1 root root      975 Nov  8  2010 anaconda-ks.cfg
-rw-r--r-- 1 root root       41 Dec  7  2010 hosts.txt                    → IP 주소 파일
-rw-r--r-- 1 root root    17973 Nov  8  2010 install.log
-rw-r--r-- 1 root root        0 Nov  8  2010 install.log.syslog
-rwxr-xr-x 1 root root     779 Feb 18  2011 ssh-key-exchange_centos.sh  → SSH Key 생성 스크립트
root:hadoop1 #>
```

이제 SSH Key를 교환할 준비가 되었다. 다음과 같이 쉘 스크립트를 실행하도록 한다. 그리고 password를 물어보면 해당 장비의 패스워드를 입력하도록 합니다. 관리를 좀 더 쉽게 하기 위해서 password를 생략할 수 있다. 하지만 권고하지는 않는다.

```
root:hadoop1 #> ./ssh-key-exchange_centos.sh
root@172.27.11.82's password: ******
Generating public/private dsa key pair.
Your identification has been saved in    /root/.ssh/id_dsa.
Your public key has been saved in    /root/.ssh/id_dsa.pub.
The key fingerprint is:
43:ed:47:62:6e:f2:b3:6a:d4:6d:a8:c0:aa:d4:87:ac    root@i-743-18415-VM
root@172.27.11.82's password: ******
id_dsa.pub
root@172.27.82.221's password: ******
Generating public/private dsa key pair.
Your identification has been saved in    /root/.ssh/id_dsa.
Your public key has been saved in /root/.ssh/id_dsa.pub.
The key fingerprint is:
16:c4:45:71:61:14:cb:90:8c:ba:ef:85:9a:a9:29:2b    root@i-743-18414-VM
root@172.27.82.221's password: ******
id_dsa.pub
root@172.27.11.82's password: ******
authorized_keys
root@172.27.82.221's password: ******
authorized_keys
```

이제 SSH Key 공유가 마무리되었다. 그러면 이제 정말 해당 호스트에 접근할 때 패스워드를 묻지 않는지 확인을 위해서 다음과 같이 입력하도록 한다.. 패스워드를 묻지 않고 OK가 떨어지면 제대로 설정한 것이다.

```
root:hadoop1 #> ssh root@hadoop2
Last login: Mon Nov 14 09:05:39 2011
[root@hadoop2 ~]#exit
...

root:hadoop1 #> ssh root@hadoop3
Last login: Mon Nov 14 09:05:39 2011
[root#hadoop3 ~]#exit
...
```

5. JDK와 Apache Hadoop 설치하기

SSH Key 교환이 마무리 되면 각 서버에 다음을 설치해야 한다. SSH Key 교환은 root 계정으로 수행하지만 Apache Hadoop의 설치는 hadoop 계정으로 진행한다.

- Sun JDK 1.6.0_37

  http://www.oracle.com/technetwork/java/javase/downloads/index.html

- Apache Hadoop 1.1.1

  http://ftp.daum.net/apache/hadoop/core/hadoop-1.1.1/hadoop-1.1.1.tar.gz

위 경로에서 JDK와 Apache Hadoop을 다운로드 하면 다음과 같이 설치할 수 있다.

```
hadoop@hadoop1:~ #> cd /usr/local/java
hadoop@hadoop1:/usr/local/java #> tar xvfz hadoop-1.1.1.tar.gz     → Apache Hadoop 설치
... 생략
hadoop@hadoop1:/usr/local/java #> chmod 755 *.bin                  → 다운로드한 JDK 실행 권한 부여
hadoop@hadoop1:/usr/local/java #> ./jdk*.bin                       → JDK 설치
```

Sun JDK와 Apache Hadoop 설치가 완료되면 환경 변수를 설정한다. 영구적으로 적용하려면 사용자의 홈 디렉터리에 .bashrc 또는 .profile 파일에 다음을 추가하도록 한다.

```
export HADOOP_HOME=/usr/local/java/hadoop-1.0.3
export PATH=$PATH:$HADOOP_HOME/bin
```

다음과 같이 Apache Hadoop의 환경 변수 설정 파일에서 설치한 JDK의 홈 디렉터리를 설정한다. JDK를 /usr/local/java/jdk1.6.0_33에 설치했다면 다음과 같이 설정한다.

```
hadoop@hadoop1:/usr/local/java/hadoop-1.0.3/conf #> vi    hadoop-env.sh

# Set Hadoop-specific environment variables here.

# The only required environment variable is JAVA_HOME.    All others are
# optional. When running a distributed configuration it    is best to
# set JAVA_HOME in this file, so that it is correctly    defined on
# remote nodes.

# The java implementation to use.
export JAVA_HOME=/usr/local/java/jdk1.6.0_33       → JDK의 경로
...
```

JDK를 설정한 후에는 다음과 같이 masters와 slaves 파일에 노드의 IP 주소를 입력한다. 현재 작업은 hadoop1 장비에서 작업을 하고 있고 이 장비는 namenode로 구성할 것이므로 마스터 노드가 된다. 따라서 현재 장비의 IP 주소인 masters 파일에는 localhost를 입력하고, slaves 파일에는 hadoop2, hadoop3의 IP 주소를 입력한다.

```
hadoop@hadoop1:/usr/local/java/hadoop-1.0.3/conf #> cat masters    → hadoop1   IP 주소(localhost)
localhost

hadoop@hadoop1:/usr/local/java/hadoop-1.0.3/conf #> cat slaves     → hadoop2, hadop3 IP 주소
172.27.11.82
172.27.82.221
```

이제 〈HADOOP_HOME〉/conf/core-site.xml 파일을 다음과 같이 작성한다. 이 파일에는 Hadoop의 파일 시스템을 관리할 네임노드의 정보를 입력한다. 종종 〈name〉 태그에 들어 있는 key값을 틀리게 입력하는 경우가 발생하여 제대로 동작하지 않는 경우가 발생하므로

〈name〉 태그의 값을 확인하도록 한다.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"    href="configuration.xsl"?>

<configuration>
  <property>
      <name>fs.default.name</name>
      <value>hdfs://<HADOOP1의 IP>:9000</value>    → hadoop 1노드의 IP 주소를 입력
  </property>
</configuration>
```

이제 〈HADOOP_HOME〉/conf/mapred-site.xml 파일을 다음과 같이 작성한다. 이 파일에는 Hadoop의 Job Tracker가 실행할 정보를 입력한다.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"    href="configuration.xsl"?>

<configuration>
  <property>
      <name>mapred.job.tracker</name>
    <value><HADOOP1의 IP>:9001</value>    → hadoop 1노드의 IP 주소를 입력
  </property>
</configuration>
```

모든 작업이 끝나면 hadoop1 노드에서 hadoop2와 hadoop3 노드로 다음의 커맨드를 이용하여 환경설정 파일을 모두 동기화 한다. Hadoop이 구동하면서 동기화 작업을 진행하기는 하지만 종종 제대로 되지 않는 경우가 발생하기도 한다. 그래서 현장에서는 다음과 같이 수동으로 반영하는 경우가 흔하게 일어난다.

```
hadoop@hadoop1:/usr/local/java/hadoop-1.0.3/conf #> scp *    root@hadoop2:/usr/local/java/hadoop-1.0.3
capacity-scheduler.xml                                      100% 7457      7.3KB/s    00:00
configuration.xsl                                           100%  535      0.5KB/s    00:00
core-site.xml                                               100%  282      0.3KB/s    00:00
fair-scheduler.xml                                          100%  327      0.3KB/s    00:00
hadoop-env.sh                                               100% 2116      2.1KB/s    00:00
hadoop-metrics2.properties                                  100% 1488      1.5KB/s    00:00
hadoop-policy.xml                                           100% 4644      4.5KB/s    00:00
hdfs-site.xml                                               100%  258      0.3KB/s    00:00
log4j.properties                                            100% 4441      4.3KB/s    00:00
```

```
mapred-queue-acls.xml                                    100% 2033     2.0KB/s   00:00
mapred-site.xml                                          100%  278     0.3KB/s   00:00
masters                                                  100%   10     0.0KB/s   00:00
slaves                                                   100%   27     0.0KB/s   00:00
ssl-client.xml.example                                   100% 1243     1.2KB/s   00:00
ssl-server.xml.example                                   100% 1195     1.2KB/s   00:00
taskcontroller.cfg                                       100%  382     0.4KB/s   00:00

hadoop@hadoop1:/usr/local/java/hadoop-1.0.3/conf #> scp *    root@hadoop3:/usr/local/java/hadoop-1.0.3
capacity-scheduler.xml                                   100% 7457     7.3KB/s   00:00
configuration.xsl                                        100%  535     0.5KB/s   00:00
core-site.xml                                            100%  282     0.3KB/s   00:00
fair-scheduler.xml                                       100%  327     0.3KB/s   00:00
hadoop-env.sh                                            100% 2116     2.1KB/s   00:00
hadoop-metrics2.properties                               100% 1488     1.5KB/s   00:00
hadoop-policy.xml                                        100% 4644     4.5KB/s   00:00
hdfs-site.xml                                            100%  258     0.3KB/s   00:00
log4j.properties                                         100% 4441     4.3KB/s   00:00
mapred-queue-acls.xml                                    100% 2033     2.0KB/s   00:00
mapred-site.xml                                          100%  278     0.3KB/s   00:00
masters                                                  100%   10     0.0KB/s   00:00
slaves                                                   100%   27     0.0KB/s   00:00
ssl-client.xml.example                                   100% 1243     1.2KB/s   00:00
ssl-server.xml.example                                   100% 1195     1.2KB/s   00:00
taskcontroller.cfg                                       100%  382     0.4KB/s   00:00
```

설정 파일의 동기화가 완료되면 분산 파일 시스템을 초기화 한다. 이 작업을 포맷팅한다고 한다.

```
hadoop@hadoop1:/usr/local/java/hadoop-1.0.3/bin #> hadoop namenode -format
11/12/07 03:00:08 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = hadoop1/172.27.135.97
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 1.1.1
STARTUP_MSG:   build =
https://svn.apache.org/repos/asf/hadoop/common/branches/branch-0.20-security-205 -r 1179940;
compiled by 'hortonfo' on Fri Oct  7 06:25:10 UTC 2011
************************************************************/
11/12/07 03:00:08 INFO util.GSet: VM type       = 64-bit
11/12/07 03:00:08 INFO util.GSet: 2% max memory =  17.77875 MB
11/12/07 03:00:08 INFO util.GSet: capacity      = 2^21 = 2097152 entries
11/12/07 03:00:08 INFO util.GSet: recommended=2097152,  actual=2097152
11/12/07 03:00:08 INFO namenode.FSNamesystem:  fsOwner=root
```

```
11/12/07 03:00:08 INFO namenode.FSNamesystem:    supergroup=supergroup
11/12/07 03:00:08 INFO namenode.FSNamesystem:    isPermissionEnabled=true
11/12/07 03:00:08 INFO namenode.FSNamesystem: dfs.block.invalidate.limit=100
11/12/07 03:00:08 INFO namenode.FSNamesystem:    isAccessTokenEnabled=false
accessKeyUpdateInterval=0 min(s),    accessTokenLifetime=0 min(s)
11/12/07 03:00:08 INFO namenode.NameNode: Caching file    names occuring more than 10 times
11/12/07 03:00:09 INFO common.Storage: Image file of size    110 saved in 0 seconds.
11/12/07 03:00:09 INFO common.Storage: Storage directory    /tmp/hadoop-hadoop/dfs/name has been
successfully formatted.
11/12/07 03:00:09 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at    172.27.135.97/172.27.135.97
************************************************************/
```

※ 위 정보에서 Storage directory가 /tmp/hadoop-hadoop/dfs/name 으로 설정된 것을 확인할 수 있다. 기본 스토리지 디렉터리는 /tmp/hadoop-${user.name}으로 설정되므로 실제 디렉터리 구조가 위와 같이 구성된 것이다. 한 가지 주의해야할 점은 <HADOOP_HOME>/conf/hdfs-site.xml 파일에 별도로 설정하지 않으면 기본으로 /tmp 밑에 스토리지 디렉터리가 구성되므로 운영을 위해서 설정한다면 이 경로를 반드시 수정해야 한다. 이 부분은 이 문서의 뒷부분에서 다시 설명한다.

모든 설정이 완료되었으므로 다음의 커맨드로 hadoop을 시작한다.

```
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1/bin #> start-all.sh
starting namenode, logging to    /var/log/hadoop/hadoop/hadoop-hadoop-namenode-hadoop1.out
172.27.11.82: starting datanode, logging to
/var/log/hadoop/hadoop/hadoop-hadoop-datanode-hadoop1.out
172.27.82.221: starting datanode, logging to
/var/log/hadoop/hadoop/hadoop-hadoop-datanode-hadoop1.out
localhost: starting secondarynamenode, logging to
/var/log/hadoop/hadoop/hadoop-hadoop-secondarynamenode-hadoop1.out
starting jobtracker, logging to    /var/log/hadoop/hadoop/hadoop-hadoop-jobtracker-i-hadoop1.out
172.27.82.221: starting tasktracker, logging to
/var/log/hadoop/hadoop/hadoop-hadoop-tasktracker-hadoop1.out
172.27.11.82: starting tasktracker, logging to
/var/log/hadoop/hadoop/hadoop-hadoop-tasktracker-hadoop1.out
```

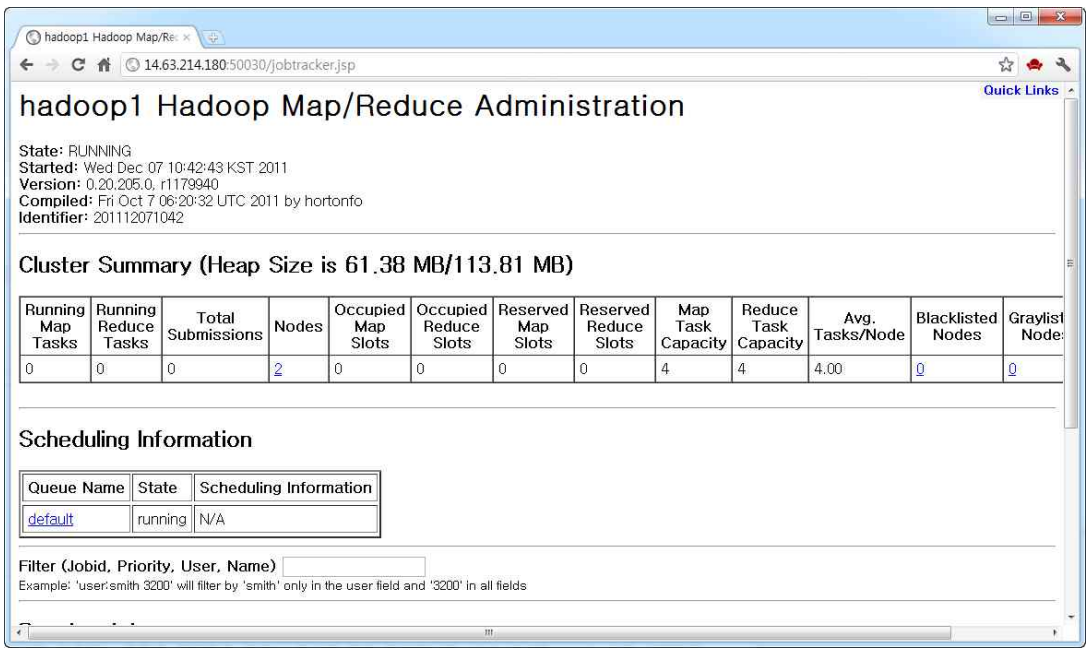
start-all.sh 쉘 스크립트를 이용하여 Hadoop을 시작하면 마스터 노드인 namenode에는

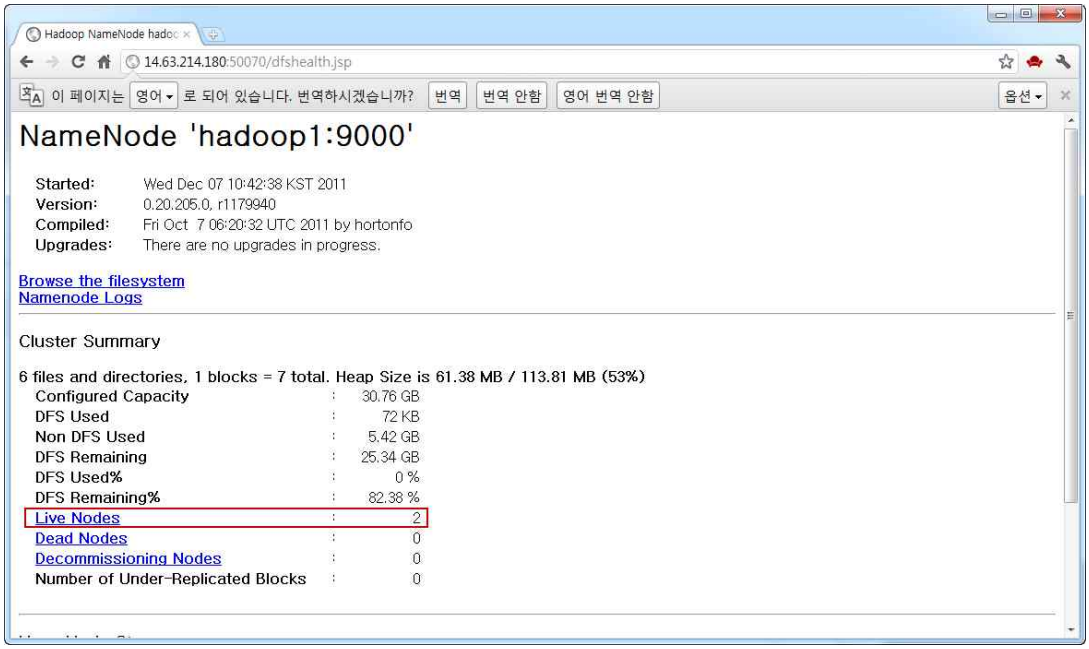
namenode, job tracker, secondary namenode가 동작해야 하며 슬레이브 노드인 datanode에는 task tracker, datanode가 동작해야 한다. 이를 확인하기 위해서 ssh 커맨드로 다음과 같이 확인해볼 수 있다.

```
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1/bin #> ssh hadoop@hadoop2 jps
27843 Jps
25411 DataNode
25522 TaskTracker
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1/bin #> ssh hadoop@hadoop3 jps
25169 TaskTracker
25069 DataNode
27512 Jps
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1/bin #> jps
32585 Jps
30180 JobTracker
29908 NameNode
30089 SecondaryNameNode
```

6. 설치 후 점검하기

Hadoop 관련 프로세스가 정상적으로 동작하고 있으므로 hadoop1 장비의 IP 주소를 웹 브라우저에서 입력하여 Job Tracker의 동작 유무를 확인한다. Job Tracker는 http://<PUBLIC_IP>:50030, HDFS는 http://<PUBLIC_IP>:50070이다.





하지만 Hadoop의 설치 과정은 매우 번잡스럽고 system engineer가 해야 하는 작업들이 많아서 작업 중 실수라도 하게 되면 미궁으로 빠질 수 있는 소지가 충분하다. 일반적으로 발생하는 증상은 다음과 같다.

○ 마스터와 슬레이브 노드의 데몬이 제대로 동작하지 않는 경우

● 마스터 노드의 namenode, secondary namenode, job tracker가 동작하지 않는 경우

● 슬레이브 노드의 task tracker, datanode가 동작하지 않는 경우

● 일부 데몬이 최초에는 구동했다가 바로 kill 되는 경우

▷ 각 노드의 환경 설정 디렉터리의 파일이 모두 동일한지 확인한다.

▷ 설치 과정 중 노드 정보가 변경되었다면 hdfs-site.xml 파일의 dfs.name.dir dfs.data.dir 디렉터리를 삭제하고 다시 HDFS를 포맷팅 한다.

○ HDFS의 파일 블록이 시간이 지나면서 corrupt 현상을 보이는 경우

→ HDFS 웹 콘솔에서 빨간색 경고창이 나타남

▷ hdfs-site.xml 파일에 데이터 및 네임 노드의 디렉터리를 별도로 지정하지 않았다면 /tmp 디렉터리로 설정되고 이 디렉터리는 OS가 임의로 삭제할 수 있

기 때문에 발생 가능성 있음. hdfs-site.xml 파일에 dfs.name.dir dfs.data.dir 경로를 수정하도록 한다.

○ 마스터 노드는 제대로 동작하는 슬레이브 노드가 동작하지 않는 경우

→ Job Tracker, HDFS 웹 콘솔에서 live nodes가 0로 나타나거나 개수가 틀린 문제

▷ 마스터 노드: 〈HADOOP_HOME〉/log 디렉터리의 로그 파일을 점검하여 마스터 노드가 슬레이브 노드를 인지하지 못하는 이유를 확인한다.

▷ 슬레이브 노드: 〈HADOOP_HOME〉/log 디렉터리의 로그 파일을 점검하여 슬레이브 노드가 마스터 노드를 인지하지 못하는 이유를 확인한다.

해결방법을 찾아봐도 해결이 되지 않으면 다시 한 번 다음의 사항을 전체적으로 점검해본다.

○ /etc/hosts 파일에 모두 동기화 되어 있는가?

○ /etc/hosts 파일에 정의되어 있는 마스터 노드와 슬레이브 노드가 각각의 노드에서 모두 접근이 되는가?

○ HDFS를 구성하는 각 노드의 디렉터리가 임시 디렉터리로 설정되어 있지 않는가? 설정되어 있다면 hdfs-site.xml 파일에 dfs.name.dir과 dfs.data.dir 경로를 변경한다.

○ core-site.xml 파일의 fs.default.name에 해당하는 값이 제대로 설정되어 있는가? key값과 value가 모두 틀리는 경우가 종종 발생하므로 이상이 없는지 점검해보도록 한다.

○ mapred-site.xml 파일의 mapred.job.tracker에 해당하는 값이 제대로 설정되어 있는가?

○ HDFS를 포맷팅 한 이후에 노드 정보가 변경되거나 또는 제대로 구동되지 않은 상태에서 환경설정 파일을 변경한 적이 있는가?

7. 테스트하기

Apache Hadoop을 설치한 이후에 다음의 커맨드를 이용하여 임시 파일을 생성해본다. 다음의 커맨드는 Hadoop에 포함되어 있는 예제 프로그램으로 임시로 10만건의 로그를 생성한다.

```
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> hadoop jar hadoop-examples-1.1.1.jar   teragen \
                                                100000 /teragen_output
Generating 100000 using 2 maps with step of 50000
```

```
11/12/07 12:09:06 INFO mapred.JobClient: Running job:  job_201112071042_0002
11/12/07 12:09:07 INFO mapred.JobClient:  map 0% reduce 0%
11/12/07 12:09:20 INFO mapred.JobClient:  map 50% reduce 0%
11/12/07 12:09:22 INFO mapred.JobClient:  map 100% reduce 0%
11/12/07 12:09:27 INFO mapred.JobClient: Job complete:  job_201112071042_0002
11/12/07 12:09:27 INFO mapred.JobClient: Counters: 19
11/12/07 12:09:27 INFO mapred.JobClient:  Job Counters
11/12/07 12:09:27 INFO mapred.JobClient:  SLOTS_MILLIS_MAPS=16596
11/12/07 12:09:27 INFO mapred.JobClient:  Total time spent by all reduces waiting  after
reserving slots (ms)=0
11/12/07 12:09:27 INFO mapred.JobClient:  Total time spent by all maps waiting  after
reserving slots (ms)=0
11/12/07 12:09:27 INFO mapred.JobClient:  Launched map tasks=2
11/12/07 12:09:27 INFO mapred.JobClient:  SLOTS_MILLIS_REDUCES=0
11/12/07 12:09:27 INFO mapred.JobClient:  File Input Format Counters
11/12/07 12:09:27 INFO mapred.JobClient:  Bytes Read=0
11/12/07 12:09:27 INFO mapred.JobClient:  File Output Format Counters
11/12/07 12:09:27 INFO mapred.JobClient:  Bytes Written=10000000
11/12/07 12:09:27 INFO mapred.JobClient:  FileSystemCounters
11/12/07 12:09:27 INFO mapred.JobClient:  HDFS_BYTES_READ=164
11/12/07 12:09:27 INFO mapred.JobClient:  FILE_BYTES_WRITTEN=42362
11/12/07 12:09:27 INFO mapred.JobClient:  HDFS_BYTES_WRITTEN=10000000
11/12/07 12:09:27 INFO mapred.JobClient:  Map-Reduce Framework
11/12/07 12:09:27 INFO mapred.JobClient:  Map input records=100000
11/12/07 12:09:27 INFO mapred.JobClient:  Physical memory (bytes)  snapshot=139001856
11/12/07 12:09:27 INFO mapred.JobClient:  Spilled Records=0
11/12/07 12:09:27 INFO mapred.JobClient:  CPU time spent (ms)=1010
11/12/07 12:09:27 INFO mapred.JobClient:  Total committed heap usage  (bytes)=128712704
11/12/07 12:09:27 INFO mapred.JobClient:  Virtual memory (bytes) snapshot=1112719360
11/12/07 12:09:27 INFO mapred.JobClient:  Map input bytes=100000
11/12/07 12:09:27 INFO mapred.JobClient:  Map output records=100000
11/12/07 12:09:27 INFO mapred.JobClient:  SPLIT_RAW_BYTES=164
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> hadoop fs -ls /teragen_output
Found 4 items
-rw-r--r--  1 root  supergroup  0 2011-12-07 12:09  /teragen_output/_SUCCESS
drwxr-xr-x  - root  supergroup  0 2011-12-07 12:09  /teragen_output/_logs
-rw-r--r--  1 root  supergroup  5000000 2011-12-07 12:09  /teragen_output/part-00000
-rw-r--r--  1 root  supergroup  5000000 2011-12-07 12:09  /teragen_output/part-00001
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> hadoop fs -cat /teragen_output/part-00000 | head
.t^#\|v$2\
0AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHH
75@~?'WdUF
1IIIIIIIIIIJJJJJJJJJKKKKKKKKKKLLLLLLLLLLMMMMMMMMMMNNNNNNNNNNOOOOOOOOOOPPPPPPPP
w[o||:N&H,
2QQQQQQQQQQRRRRRRRRRRSSSSSSSSSSTTTTTTTTTTUUUUUUUUUUVVVVVVVVVVWWWWWWWWWWXXXXXXXX
```

```
^Eu)<n#kdP
3YYYYYYYYYYZZZZZZZZZZAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFF
+l-$$OE/ZH
4GGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLLLLMMMMMMMMMMNNNNNNNN
LsS8)|.ZLD
5OOOOOOOOOOPPPPPPPPPPQQQQQQQQQQRRRRRRRRRRSSSSSSSSSSTTTTTTTTTTUUUUUUUUUUVVVVVVVV
le5awB.$sm
6WWWWWWWWWWXXXXXXXXXXYYYYYYYYYYZZZZZZZZZZAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDD
q__[fwhKFg
7EEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLL
;L+!2rT~hd
8MMMMMMMMMMNNNNNNNNNNOOOOOOOOOOPPPPPPPPPPQQQQQQQQQQRRRRRRRRRRSSSSSSSSSSTTTTTTTT
M^*dDE;6^<
9UUUUUUUUUUVVVVVVVVVVWWWWWWWWWWXXXXXXXXXXYYYYYYYYYYZZZZZZZZZZAAAAAAAAAABBBBBBBB
```

teragen을 이용하여 생성한 파일은 terasort를 이용하여 정렬할 수 있다. 다음의 커맨드는 teragen의 출력 파일을 입력 파일로 사용하여 정렬하는 terasort를 실행한다.

```
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> hadoop jar hadoop-examples-1.1.1.jar    terasort \
                                                /teragen_output /terasort_output
11/12/07 12:14:18 INFO terasort.TeraSort: starting
11/12/07 12:14:19 INFO mapred.FileInputFormat: Total    input paths to process : 2
11/12/07 12:14:19 WARN util.NativeCodeLoader: Unable to    load native-hadoop library for your
platform... using builtin-java classes    where applicable
11/12/07 12:14:19 INFO compress.CodecPool: Got brand-new    compressor
Making 1 from 100000 records
Step size is 100000.0
11/12/07 12:14:20 INFO mapred.FileInputFormat: Total    input paths to process : 2
11/12/07 12:14:20 INFO mapred.JobClient: Running job:    job_201112071042_0004
11/12/07 12:14:21 INFO mapred.JobClient:   map 0% reduce 0%
11/12/07 12:14:38 INFO mapred.JobClient:   map 50% reduce 0%
11/12/07 12:14:39 INFO mapred.JobClient:   map 100% reduce 0%
11/12/07 12:14:53 INFO mapred.JobClient:   map 100% reduce 100%
11/12/07 12:14:58 INFO mapred.JobClient: Job complete:    job_201112071042_0004
11/12/07 12:14:58 INFO mapred.JobClient: Counters: 30
11/12/07 12:14:58 INFO mapred.JobClient:    Job Counters
11/12/07 12:14:58 INFO mapred.JobClient:      Launched reduce tasks=1
11/12/07 12:14:58 INFO mapred.JobClient:      SLOTS_MILLIS_MAPS=22548
11/12/07 12:14:58 INFO mapred.JobClient:      Total time spent by all reduces waiting    after
reserving slots (ms)=0
11/12/07 12:14:58 INFO mapred.JobClient:      Total time spent by all maps waiting    after
reserving slots (ms)=0
11/12/07 12:14:58 INFO mapred.JobClient:      Launched map tasks=2
11/12/07 12:14:58 INFO mapred.JobClient:      Data-local map tasks=2
```

```
11/12/07 12:14:58 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=13723
11/12/07 12:14:58 INFO mapred.JobClient:   File Input Format Counters
11/12/07 12:14:58 INFO mapred.JobClient:     Bytes Read=100000000
11/12/07 12:14:58 INFO mapred.JobClient:   File Output Format Counters
11/12/07 12:14:58 INFO mapred.JobClient:     Bytes Written=100000000
11/12/07 12:14:58 INFO mapred.JobClient:   FileSystemCounters
11/12/07 12:14:58 INFO mapred.JobClient:     FILE_BYTES_READ=204000294
11/12/07 12:14:58 INFO mapred.JobClient:     HDFS_BYTES_READ=100000202
11/12/07 12:14:58 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=306066259
11/12/07 12:14:58 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=100000000
11/12/07 12:14:58 INFO mapred.JobClient:   Map-Reduce Framework
11/12/07 12:14:58 INFO mapred.JobClient:     Map output materialized bytes=102000012
11/12/07 12:14:58 INFO mapred.JobClient:     Map input records=1000000
11/12/07 12:14:58 INFO mapred.JobClient:     Reduce shuffle bytes=102000012
11/12/07 12:14:58 INFO mapred.JobClient:     Spilled Records=3000000
11/12/07 12:14:58 INFO mapred.JobClient:     Map output bytes=100000000
11/12/07 12:14:58 INFO mapred.JobClient:     Total committed heap usage   (bytes)=447021056
11/12/07 12:14:58 INFO mapred.JobClient:     CPU time spent (ms)=13240
11/12/07 12:14:58 INFO mapred.JobClient:     Map input bytes=100000000
11/12/07 12:14:58 INFO mapred.JobClient:     SPLIT_RAW_BYTES=202
11/12/07 12:14:58 INFO mapred.JobClient:     Combine input records=0
11/12/07 12:14:58 INFO mapred.JobClient:     Reduce input records=1000000
11/12/07 12:14:58 INFO mapred.JobClient:     Reduce input groups=1000000
11/12/07 12:14:58 INFO mapred.JobClient:     Combine output records=0
11/12/07 12:14:58 INFO mapred.JobClient:     Physical memory (bytes)   snapshot=482889728
11/12/07 12:14:58 INFO mapred.JobClient:     Reduce output records=1000000
11/12/07 12:14:58 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=1663201280
11/12/07 12:14:58 INFO mapred.JobClient:     Map output records=1000000
11/12/07 12:14:58 INFO terasort.TeraSort: done

hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> hadoop fs -cat /terasort_output/part-00000 | head
    ]:/sW_w
499209UUUUUUUUUUVVVVVVVVVVWWWWWWWWWWXXXXXXXXXXYYYYYYYYYYZZZZZZZZZZAAAAAAAAAABBBBBBBB
    lERv6<F
821056QQQQQQQQQQRRRRRRRRRRSSSSSSSSSSTTTTTTTTTTUUUUUUUUUUVVVVVVVVVVWWWWWWWWWWXXXXXXXX
    !1?/A^=o
259481IIIIIIIIIIJJJJJJJJJKKKKKKKKKKLLLLLLLLLLMMMMMMMMMMNNNNNNNNNNOOOOOOOOOOPPPPPPPP
    !;_ kUwQ
605958QQQQQQQQQQRRRRRRRRRRSSSSSSSSSSTTTTTTTTTTUUUUUUUUUUVVVVVVVVVVWWWWWWWWWWXXXXXXXX
    !lxDS~}%
314981GGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJKKKKKKKKKKLLLLLLLLLLMMMMMMMMMMNNNNNNNN
    ">R<iMA:
951912AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHH
    #!Zp?vPG
605042UUUUUUUUUUVVVVVVVVVVWWWWWWWWWWXXXXXXXXXXYYYYYYYYYYZZZZZZZZZZAAAAAAAAAABBBBBBBB
```

```
  #TT8\FaI
344853QQQQQQQQQQRRRRRRRRRRSSSSSSSSSSTTTTTTTTTTUUUUUUUUUUVVVVVVVVVVWWWWWWWWWWXXXXXXXX
  $U._,/WO
505847GGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLLLLMMMMMMMMMMNNNNNNNN
  %/P{/=@9
297497OOOOOOOOOOPPPPPPPPPPQQQQQQQQQQRRRRRRRRRRSSSSSSSSSSTTTTTTTTTTUUUUUUUUUUVVVVVVVV
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> hadoop fs -cat /terasort_1M/part-00000 |  tail
  ~~u|0<~+q/
854770EEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLL
  ~~xkL'M=mR
869627OOOOOOOOOOPPPPPPPPPPQQQQQQQQQQRRRRRRRRRRSSSSSSSSSSTTTTTTTTTTUUUUUUUUUUVVVVVVVV
  ~~yYyt'^G(
340244MMMMMMMMMMNNNNNNNNNNOOOOOOOOOOPPPPPPPPPPQQQQQQQQQQRRRRRRRRRRSSSSSSSSSSTTTTTTTT
  ~~ys|954d3
967786IIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLLLLMMMMMMMMMMNNNNNNNNNNOOOOOOOOOOPPPPPPPP
  ~~yv{$rhqD
293961OOOOOOOOOOPPPPPPPPPPQQQQQQQQQQRRRRRRRRRRSSSSSSSSSSTTTTTTTTTTUUUUUUUUUUVVVVVVVV
  ~~{Mo]qhau
513467WWWWWWWWWWXXXXXXXXXXYYYYYYYYYYZZZZZZZZZZAAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDD
  ~~|8EMb!Y5
231277EEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLL
  ~~}FSzNAVC
226607GGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLLLLMMMMMMMMMMNNNNNNNN
  ~~}|s?E^=r
325725CCCCCCCCCCDDDDDDDDDDEEEEEEEEEEFFFFFFFFFFGGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJ
  ~~~**MtNYi
814285GGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJKKKKKKKKKKLLLLLLLLLLMMMMMMMMMMNNNNNNNN
```

Apache Hadoop에 포함되어 있는 기본 예제에는 wordcount도 포함되어 있다. wordcount를 실행하기 위해서 먼저 입력 경로로 사용할 임시 경로를 다음과 같이 생성하고 파일을 업로드한 후에 wordcount를 실행한다.

```
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> hadoop fs -mkdir /wordcount_input
                                                  → HDFS에 디렉터리 생성
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> hadoop fs -put *.txt /wordcount_input
                                                  → .txt 파일  업로드
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> hadoop jar hadoop-examples-1.1.1.jar  wordcount \
                                    /wordcount_input /wordcount_output  → Hadoop Job 실행
11/12/07 12:05:25 INFO input.FileInputFormat: Total input    paths to process : 4
11/12/07 12:05:25 INFO mapred.JobClient: Running job:    job_201112071042_0001
11/12/07 12:05:26 INFO mapred.JobClient:   map 0% reduce 0%
11/12/07 12:05:39 INFO mapred.JobClient:   map 50% reduce 0%
11/12/07 12:05:40 INFO mapred.JobClient:   map 100% reduce 0%
11/12/07 12:05:51 INFO mapred.JobClient:   map 100% reduce 100%
```

```
11/12/07 12:05:56 INFO mapred.JobClient: Job complete:  job_201112071042_0001
11/12/07 12:05:56 INFO mapred.JobClient: Counters: 29
11/12/07 12:05:56 INFO mapred.JobClient:   Job Counters
11/12/07 12:05:56 INFO mapred.JobClient:     Launched reduce tasks=1
11/12/07 12:05:56 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=27769
11/12/07 12:05:56 INFO mapred.JobClient:     Total time spent by all reduces waiting  after
reserving slots (ms)=0
11/12/07 12:05:56 INFO mapred.JobClient:     Total time spent by all maps waiting  after
reserving slots (ms)=0
11/12/07 12:05:56 INFO mapred.JobClient:     Launched map tasks=4
11/12/07 12:05:56 INFO mapred.JobClient:     Data-local map tasks=4
11/12/07 12:05:56 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=10247
11/12/07 12:05:56 INFO mapred.JobClient:   File Output Format Counters
11/12/07 12:05:56 INFO mapred.JobClient:     Bytes Written=161360
11/12/07 12:05:56 INFO mapred.JobClient:   FileSystemCounters
11/12/07 12:05:56 INFO mapred.JobClient:     FILE_BYTES_READ=215086
11/12/07 12:05:56 INFO mapred.JobClient:     HDFS_BYTES_READ=449207
11/12/07 12:05:56 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=537349
11/12/07 12:05:56 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=161360
11/12/07 12:05:56 INFO mapred.JobClient:   File Input Format Counters
11/12/07 12:05:56 INFO mapred.JobClient:     Bytes Read=448761
11/12/07 12:05:56 INFO mapred.JobClient:   Map-Reduce Framework
11/12/07 12:05:56 INFO mapred.JobClient:     Map output materialized bytes=215104
11/12/07 12:05:56 INFO mapred.JobClient:     Map input records=11145
11/12/07 12:05:56 INFO mapred.JobClient:     Reduce shuffle bytes=215104
11/12/07 12:05:56 INFO mapred.JobClient:     Spilled Records=25710
11/12/07 12:05:56 INFO mapred.JobClient:     Map output bytes=645580
11/12/07 12:05:56 INFO mapred.JobClient:     CPU time spent (ms)=3140
11/12/07 12:05:56 INFO mapred.JobClient:     Total committed heap usage  (bytes)=702152704
11/12/07 12:05:56 INFO mapred.JobClient:     Combine input records=57865
11/12/07 12:05:56 INFO mapred.JobClient:     SPLIT_RAW_BYTES=446
11/12/07 12:05:56 INFO mapred.JobClient:     Reduce input records=12855
11/12/07 12:05:56 INFO mapred.JobClient:     Reduce input groups=12497
11/12/07 12:05:56 INFO mapred.JobClient:     Combine output records=12855
11/12/07 12:05:56 INFO mapred.JobClient:     Physical memory (bytes)  snapshot=756113408
11/12/07 12:05:56 INFO mapred.JobClient:     Reduce output records=12497
11/12/07 12:05:56 INFO mapred.JobClient:     Virtual memory (bytes)  snapshot=2737614848
11/12/07 12:05:56 INFO mapred.JobClient:     Map output records=57865
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> hadoop fs -ls /
Found 3 items
drwxr-xr-x   - root   supergroup          0 2011-12-07 10:04   /tmp
drwxr-xr-x   - root   supergroup          0 2011-12-07 12:05   /wordcount_input
drwxr-xr-x   - root   supergroup          0 2011-12-07 12:05   /wordcount_output → 결과 디렉터리
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> hadoop fs -ls /wordcount_output
Found 3 items
-rw-r--r--   1 root   supergroup          0 2011-12-07 12:05   /wordcount_output/_SUCCESS
```

```
drwxr-xr-x   - root   supergroup          0 2011-12-07 12:05   /wordcount_output/_logs
-rw-r--r--   1 root   supergroup     161360 2011-12-07 12:05   /wordcount_output/part-r-00000
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> hadoop fs -cat /wordcount_output/part-r-00000 \
                                                | more            → 결과 파일 확인
"*".     1
"+".     1
"-"      2
"."      3
".."     1
"..",    1
".Trash"          1
"/"      2
"/".     2
"0.0.0.0"         1
";"      1
"AS      3
"Bad     2
"Browse  1
...
```

wordcoun나 terasort를 한 결과는 HDFS 상에 존재하므로 이 결과물을 로컬 파일 시스템으로 가져와서 타 시스템으로 전달하려면 다음과 같이 작업한다.

```
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> hadoop fs -getmerge /terasort_output  \
                                                  terasort_output.txt → 로컬로 파일 내리기
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> cat terasort_output.txt  → 로컬 파일 보기
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> hadoop fs -get /teragen_output .
                                                                  → 로컬로 디렉터리 내리기
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> ls
bin             etc                                ivy             sbin
build.xml       hadoop-ant-1.1.1.jar          ivy.xml       share
c++               hadoop-core-1.1.1.jar         lib             src
CHANGES.txt     hadoop-examples-1.1.1.jar     libexec       teragen_output
conf            hadoop-test-1.1.1.jar         LICENSE.txt   webapps
contrib         hadoop-tools-1.1.1.jar        NOTICE.txt
docs            include                            README.txt
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> cd teragen_output
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1/teragen_output #> ls -lsa
total 9796
drwxr-xr-x 3 root root       4096 Dec  7 12:20 _logs
-rw-r--r-- 1 root root 5000000 Dec  7 12:20 part-00000
-rw-r--r-- 1 root root 5000000 Dec  7 12:20 part-00001
-rw-r--r-- 1 root root       0 Dec  7 12:20 _SUCCESS
```

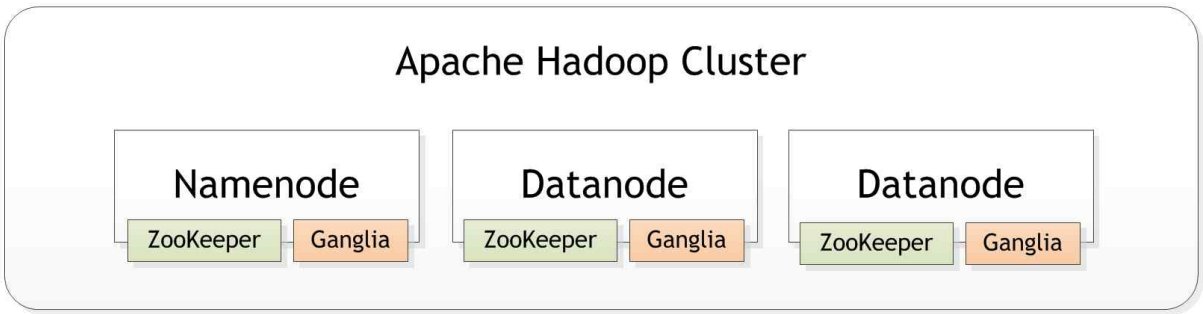
```
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1/teragen_output #> head  part-00000 → 파일 내용 보기
.t^#\|v$2\
0AAAAAAAAAABBBBBBBBBCCCCCCCCCDDDDDDDDDEEEEEEEEEFFFFFFFFFGGGGGGGGGHHHHHHHH
75@~?'WdUF
1IIIIIIIIIJJJJJJJJJKKKKKKKKKLLLLLLLLLMMMMMMMMMNNNNNNNNNOOOOOOOOOPPPPPPPP
w[o||:N&H,
2QQQQQQQQQRRRRRRRRRSSSSSSSSSTTTTTTTTTUUUUUUUUUVVVVVVVVVWWWWWWWWWXXXXXXXX
^Eu)<n#kdP
3YYYYYYYYYZZZZZZZZZAAAAAAAAAABBBBBBBBBCCCCCCCCCDDDDDDDDDEEEEEEEEEFFFFFFFF
+l-$$OE/ZH
4GGGGGGGGGHHHHHHHHHIIIIIIIIIJJJJJJJJJKKKKKKKKKLLLLLLLLLMMMMMMMMMNNNNNNNN
LsS8)|.ZLD
5OOOOOOOOOPPPPPPPPPQQQQQQQQQRRRRRRRRRSSSSSSSSSTTTTTTTTTUUUUUUUUUVVVVVVVV
le5awB.$sm
6WWWWWWWWWXXXXXXXXXYYYYYYYYYZZZZZZZZZAAAAAAAAABBBBBBBBBCCCCCCCCCDDDDDDDD
q__[fwhKFg
7EEEEEEEEEFFFFFFFFFGGGGGGGGGHHHHHHHHHIIIIIIIIIJJJJJJJJJKKKKKKKKKLLLLLLLL
;L+!2rT~hd
8MMMMMMMMMNNNNNNNNNOOOOOOOOOPPPPPPPPPQQQQQQQQQRRRRRRRRRSSSSSSSSSTTTTTTTT
M^*dDE;6^<
9UUUUUUUUUVVVVVVVVVWWWWWWWWWXXXXXXXXXYYYYYYYYYZZZZZZZZZAAAAAAAAABBBBBBBB
hadoop@hadoop1:/usr/local/java/hadoop-1.1.1 #> hadoop fs -rmr /tera*  → 파일 및 디렉터리 지우기
Deleted hdfs://hadoop1:9000/teragen_output
Deleted hdfs://hadoop1:9000/teragen_output_1M
Deleted hdfs://hadoop1:9000/terasort_1M
```

Appendix :: 제2절 Apache Hive 설치 가이드

본 문서는 중소기업과 같이 인력을 보유하지 않은 조직에서 Apache Hive를 도입하기 위한 첫 번째 관문인 설치 작업을 실제 현장의 경험자의 경험과 노하우를 바탕으로 설치할 수 있도록 지침을 제공하는 것을 목적으로 한다.

1. 시스템 구성

본 가이드는 다음과 같이 이미 Hadoop Cluster가 구축되어 있으며 기본 운영체제는 CentOS를 사용하고, Hadoop Cluster는 네임노드와 데이터노드로 구성되어 있고 모든 노드는 동일한 시스템 구성과 설정 정보를 가지고 있다고 가정한다.



Apache Hive는 ANSI SQL 기반의 Data Warehouse를 위한 Big Data 솔루션으로 Hadoop Ecosystem의 일부로써 많은 개발자 및 현업 담당자들이 선호하는 도구이다. 기본적으로 Hive가 동작하려면 별도의 메타 데이터를 저장할 스토어가 필요하며 본 가이드에서는 MySQL을 기본 메타 스토어로 활용한다.

2. MySQL 설치 및 설정하기

Hadoop 설치를 위해서 우선 MySQL을 설정해야 한다. 다음의 파일을 /root/mysql_permission.sql 파일로 저장한다. 이 과정의 핵심은 ROOT 사용자에게 테이블을

생성하거나 삭제하는 등의 모든 권한을 부여하는 것이다.

```
GRANT ALL PRIVILEGES ON *.* TO root@"%" IDENTIFIED BY "";
flush privileges;
```

MySQL을 설치하지 않은 경우 다음과 같이 설치를 시도한다.

```
root@i-743-18413-VM:~ #> yum -y install mysql mysql-server mysql-devel
...
```

이제 이 파일을 MySQL에 적용시켜 ROOT 계정이 모든 권한을 가지도록 설정한다.

```
root@i-743-18413-VM:~ #> cd /etc/init.d
root@i-743-18413-VM:/etc/init.d #> ./mysqld start
Initializing MySQL database:  Installing MySQL system tables...
OK
Filling help tables...
OK

To start mysqld at boot time you have to copy
support-files/mysql.server to the right place for your system

PLEASE REMEMBER TO SET A PASSWORD FOR THE MySQL root USER !
To do so, start the server, then issue the following commands:
/usr/bin/mysqladmin -u root password 'new-password'
/usr/bin/mysqladmin -u root -h i-743-18413-VM password 'new-password'

Alternatively you can run:
/usr/bin/mysql_secure_installation

which will also give you the option of removing the test
databases and anonymous user created by default.  This is
strongly recommended for production servers.

See the manual for more instructions.

You can start the MySQL daemon with:
cd /usr ; /usr/bin/mysqld_safe &

You can test the MySQL daemon with mysql-test-run.pl
cd mysql-test ; perl mysql-test-run.pl
```

```
Please report any problems with the /usr/bin/mysqlbug script!

The latest information about MySQL is available on the web at
http://www.mysql.com
Support MySQL by buying support/licenses at http://shop.mysql.com
                                                          [  OK  ]
Starting MySQL:                                            [  OK  ]

→ ROOT의 홈 디렉터리에 저장되어 있는 .sql 파일을 다음과 같이 적용한다.

root@i-743-18413-VM:/etc/init.d #> cd
root@i-743-18413-VM:~ #> mysql -uroot -p < mysql_permission.sql
Enter password:          → 패스워드 없음
```

Hive를 사용하기 위해서 MySQL 설치 후 root 계정의 패스워드를 다음과 같이 변경한다.

```
root@i-743-18413-VM:~ #> mysql -u root -p mysql
password:
...
mysql> update user set password = password('root') where user = 'root';
mysql> flush privileges;
```

3. Apache Hive 설치 및 설정하기

Apache Hive를 압축을 풀고 다음과 같이 설정한다. 기본 설정 파일은 모두 .template 확장자를 가지므로 파일명을 변경하지 않고 파일의 사본을 사용한다.

```
root@i-743-18413-VM:~ #> wget http://ftp.daum.net/apache/hive/hive-0.9.0/hive-0.9.0.tar.gz
root@i-743-18413-VM:~ #> tar xvfz hive-0.9.0.tar.gz
root@i-743-18413-VM:~ #> ln -s hive-0.9.0 hive
root@i-743-18413-VM:~ #> cd hive/lib
root@i-743-18413-VM:~ #> wget \
                    http://www.openflamingo.org/download/hands_on_lab/mysql-connector-java-5.1.11.jar
root@i-743-18413-VM:~ #> cd /root/hive/conf
root@i-743-18413-VM:~ #> cp hive-env.sh.template hive-env.sh
root@i-743-18413-VM:~ #> vi hive-site.xml
```

이제 hive-env.sh 파일에 다음의 항목을 수정하여 JVM Heap Size를 조정하도록 한다. 만약 Hive 커맨드가 많은 JVM Heap Size를 사용해야 한다면 다시 늘리도록 조정한다.

export HADOOP_HEAPSIZE=1024 → export HADOOP_HEAPSIZE=256

hive-site.xml 파일을 다음과 같이 작성한다.

```
<configuration>
<property>
  <name>hive.metastore.local</name>
  <value>true</value>
  <description>controls whether to connect to remove metastore server or open a new metastore server in Hive Client JVM</description>
</property>

<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:mysql://localhost:3306/hive?createDatabaseIfNotExist=true</value>
  <description>JDBC connect string for a JDBC metastore</description>
</property>

<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
  <value>com.mysql.jdbc.Driver</value>
  <description>Driver class name for a JDBC metastore</description>
</property>

<property>
  <name>javax.jdo.option.ConnectionUserName</name>
  <value>root</value>
  <description>username to use against metastore database</description>
</property>

<property>
  <name>javax.jdo.option.ConnectionPassword</name>
  <value>root</value>
  <description>password to use against metastore database</description>
</property>
</configuration>
```

※ javax.jdo.option.ConnectionPassword에는 지정한 MySQL 사용자의 패스워드를 입력하는데 기본으로 MySQL에 설정되어 있는 root 계정은 패스워드가 설정되어 있지 않다. 따라서 기본 설정 그대로 사용하는 경우 이 값을 텅빈 값으로 놔두어야 하는데 이때 Hive가 패스워드를 인지하지 못하여 MySQL을 메타스토어로 사용할 수

없게 된다. 이런 이유로 MySQL root 사용자의 패스워드를 이전 단계에서 변경하였다.

Domain Server가 준비되어 있지 않은 환경에서는 Hive가 호스트명을 resolve하려는 시도를 하기 때문에 Hive가 정상적으로 동작하지 않을 수 있다. 이 문제를 해결하기 위해서 〈HADOOP_HOME〉/conf/core-site.xml 파일에 fs.default.name의 값에서 IP 주소를 마스터 장비의 호스트명으로 변경하도록 한다. 별다른 문제가 없다면 그대로 사용해도 무방하다.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://192.168.0.1:9000</value>
  </property>
</configuration>
```

이제 Hive가 설치되어 있는 곳으로 이동하여 Hive를 실행해보도록 한다.

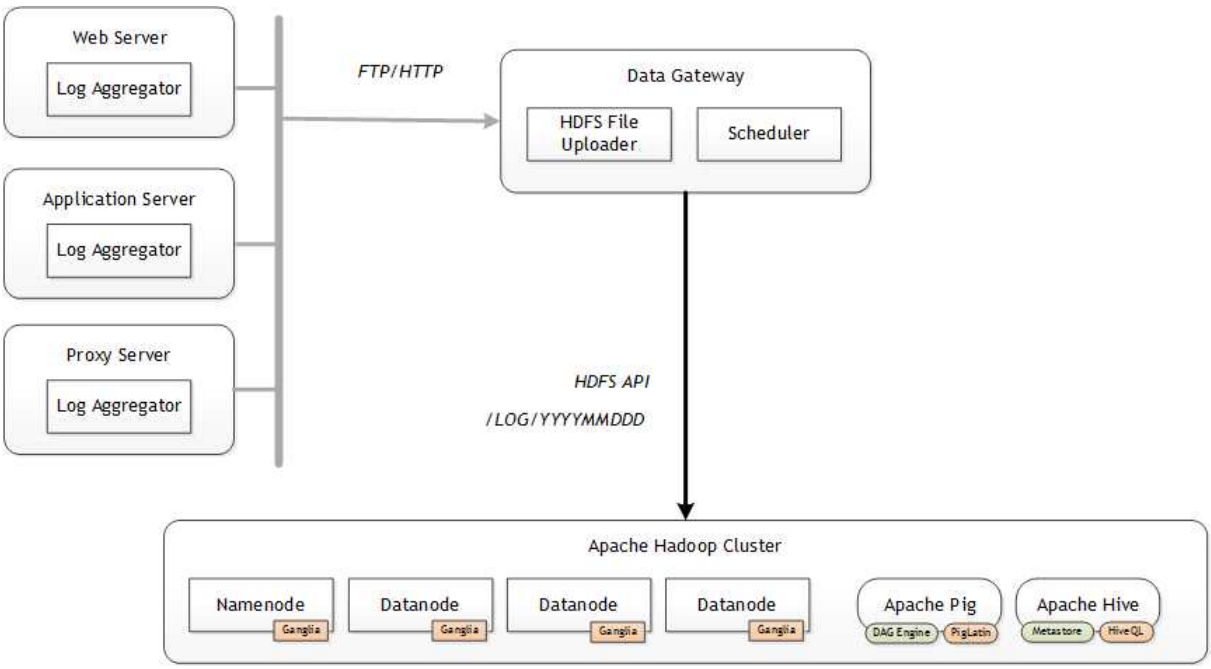
```
root@i-743-18413-VM:~/hive/bin #> cd /root/hive/bin
root@i-743-18413-VM:~/hive/bin #> hive
WARNING: org.apache.hadoop.metrics.jvm.EventCounter is deprecated. Please use
org.apache.hadoop.log.metrics.EventCounter in all the log4j.properties files.
Hive history file=/tmp/root/hive_job_log_root_201112072245_1826681426.txt
hive> show tables;
OK
Time taken: 2.313 seconds
hive> CREATE TABLE rating (userid INT, movieid INT, rating FLOAT, ds STRING) ROW FORMAT DELIMITED
FIELDS TERMINATED BY '^' STORED AS TEXTFILE;   → 평가점수 테이블 생성
OK
Time taken: 0.137 seconds
hive> show tables;
OK
rating
Time taken: 0.092 seconds
hive> describe rating;
OK
userid  string
movieid string
rating  int
```

```
ds string
Time taken: 0.06 seconds
hive> LOAD DATA INPATH '/movielens/ratings.dat' OVERWRITE INTO TABLE rating; → ratings.dat 파일을 로딩
hive> select count(*) from rating r;   → 총 건수
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201112071042_0029, Tracking URL =
http://hadoop1:50030/jobdetails.jsp?jobid=job_201112071042_0029
Kill Command = /root/hadoop-0.20.205.0/libexec/../bin/hadoop job  -Dmapred.job.tracker=172.27.135.97:9001
-kill job_201112071042_0029
2011-12-07 23:13:30,376 Stage-1 map = 0%,  reduce = 0%
2011-12-07 23:13:36,401 Stage-1 map = 50%,  reduce = 0%
2011-12-07 23:13:37,409 Stage-1 map = 100%,  reduce = 0%
2011-12-07 23:13:45,453 Stage-1 map = 100%,  reduce = 33%
2011-12-07 23:13:48,467 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201112071042_0029
OK
10000054  → 총 건수
Time taken: 31.591 seconds
hive> select r.userid from rating r where r.userid = '100';  → 사용자의 평가점수 조회
```

Appendix :: 제3절 Flamingo HDFS File Uploader 설치 가이드

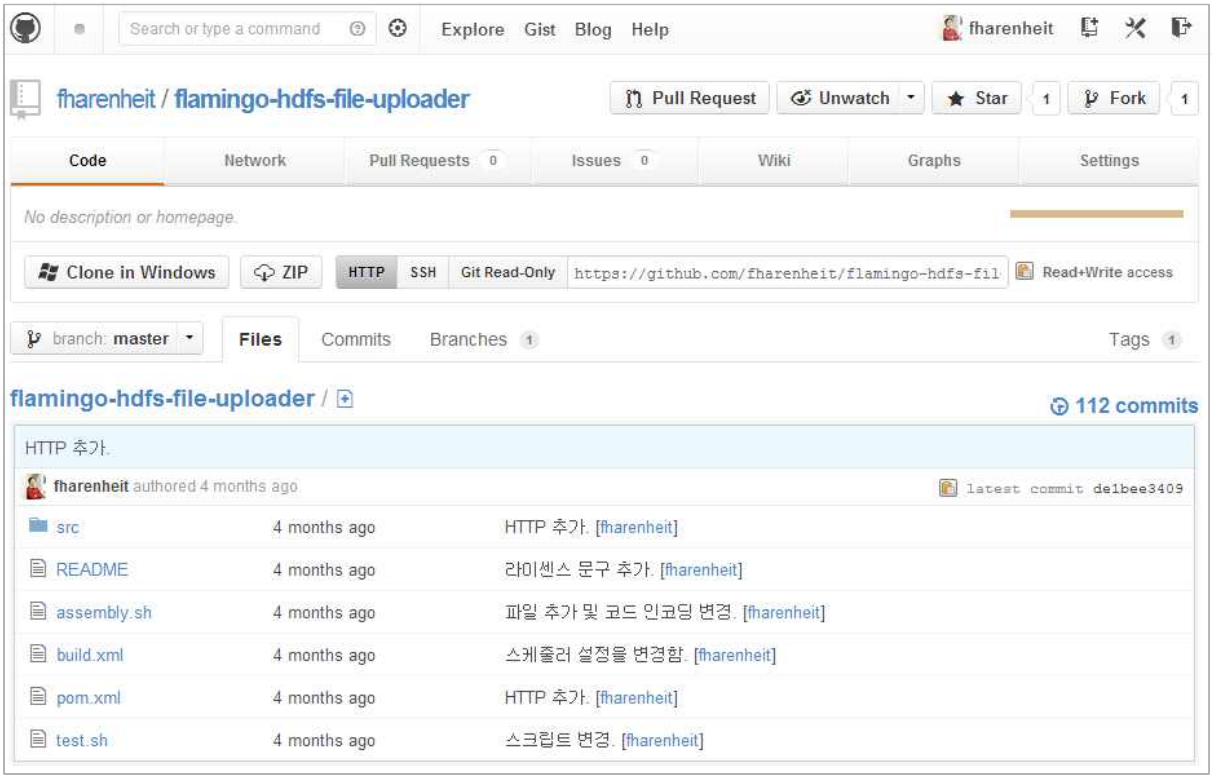
본 문서는 중소기업과 같이 인력을 보유하지 않은 조직에서 파일을 수집하여 HDFS로 업로드하는 HDFS File Uploader를 어떻게 사용할 수 있는지 지침을 제공하는 것을 목적으로 한다.

본 가이드는 다음과 같이 이미 Hadoop Cluster가 구축되어 있다는 가정하에서 설명한다. Hadoop Cluster는 네임노드와 데이터노드로 구성되어 있고 모든 노드는 동일한 시스템 구성과 설정 정보를 가지고 있다고 가정한다.

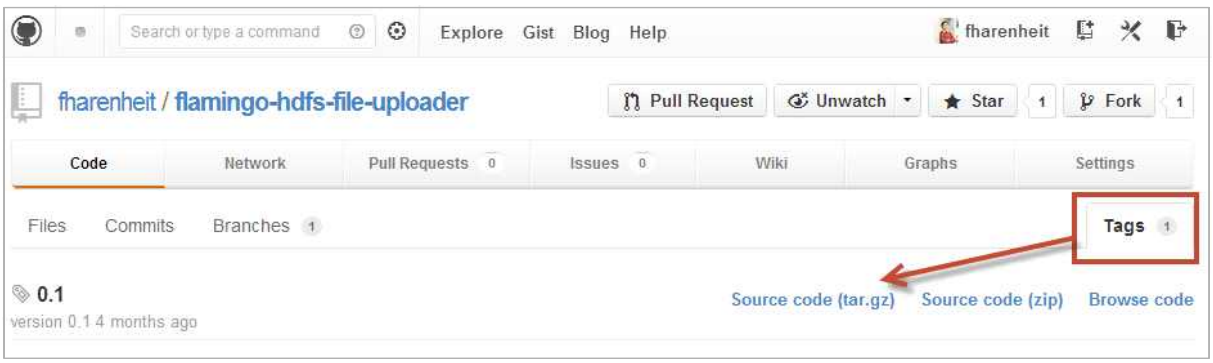


HDFS File Uploader는 Dat Gateway로 명명한 시스템의 로컬 파일 시스템에 외부 서버가 업로드한 로그 파일을 주기적으로 Hadoop Cluster의 HDFS에 파일을 업로드하는 기능을 제공한다. 본 워킹그룹에서 구현한 HDFS File Uploader는 Github에 GPLv3로 공개되어 있으며 https://github.com/fharenheit/flamingo-hdfs-file-uploader 홈페이지에서 관련 소스코드를

확인할 수 있다.



현재 개발중인 소스코드가 아닌 0.1 릴리즈 버전의 소스코드를 다운로드하려면 Github에서 다음과 같이 Tags를 눌러서 소스코드를 다운로드하도록 한다.



만약 최신 소스코드를 빌드하려면 Github에서 소스코드를 clone하기 위해서 다음의 커맨드를 입력하도록 한다.

```
#git clone https://github.com/fharenheit/flamingo-hdfs-file-uploader.git
Cloning into 'flamingo-hdfs-file-uploader'...
remote: Counting objects: 1404, done.
remote: Compressing objects: 100% (359/359), done.
remote: Total 1404 (delta 782), reused 1404 (delta 782)
Receiving objects: 100% (1404/1404), 292.41 KiB | 180 KiB/s, done.
Resolving deltas: 100% (782/782), done.
#cd flamingo-hdfs-file-uploader
#sh assembly.sh
[INFO] ------------------------------------------------------------------------
[INFO] Building Flamingo HDFS File Uploader 0.2-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
... 생략
[INFO] Building tar:
/home/user/flamingo-hdfs-file-uploader/target/flamingo-hdfs-file-uploader-0.2-SNAPSHOT-bin.tar.g
z   → 빌드 최종 파일
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 7.585s
[INFO] Finished at: Thu Dec 27 21:20:34 KST 2012
[INFO] Final Memory: 24M/246M
[INFO] ------------------------------------------------------------------------
```

소스코드 빌드가 완료되면 다음의 커맨드를 이용하여 HDFS File Uploader를 실행한다.

```
#tar xvfz flamingo-hdfs-file-uploader-0.2-SNAPSHOT-bin.tar.gz
...

#cd flamingo-hdfs-file-uploader-0.2-SNAPSHOT/
#ls -lsa
합계 52
 0 drwxrwxr-x+ 1 USER 없음     0 12월 27 21:26 .
 4 drwxrwxr-x+ 1 USER 없음     0 12월 27 21:26 ..
 4 drwxrwxr-x+ 1 USER 없음     0 12월 27 21:26 conf
40 drwxrwxr-x+ 1 USER 없음     0 12월 27 21:20 lib
 4 -rw-rw-r--  1 USER 없음 1732 12월 27 21:19 start.sh

#ls conf
applicationContext.xml   config.properties   example.xml   flamingo-uploader-1.0.xsd   log4j.xml

#cat example.xml
<?xml version="1.0" encoding="UTF-8"?>
```

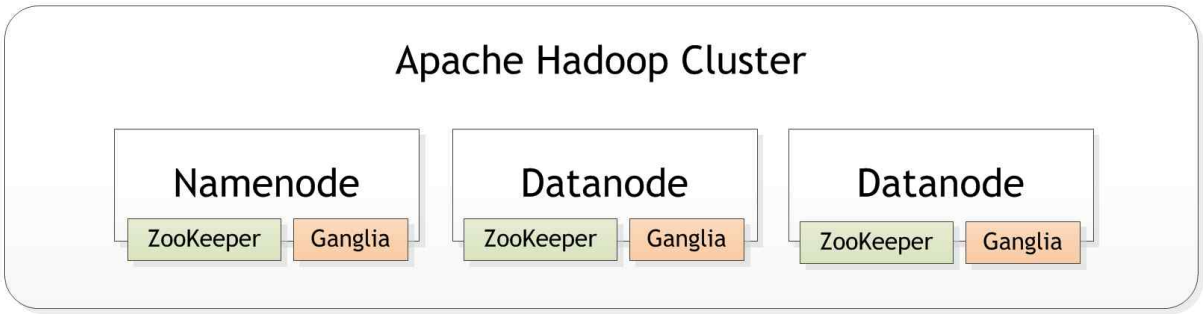
```
<flamingo xmlns="http://www.openflamingo.org/schema/uploader"
          xsi:schemaLocation="http://www.openflamingo.org/schema/uploader
flamingo-uploader-1.0.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <description>서울시 공공 데이터 배치 업로드</description>

  <clusters>
      <cluster name="dev" description="개발 클러스터">
          <fs.default.name>hdfs://172.27.21.143:9000</fs.default.name>
          <mapred.job.tracker>172.27.21.143:9001</mapred.job.tracker>
      </cluster>
  </clusters>

  <globalVariables>
      <globalVariable name="currentDate" value="${dateFormat('yyyyMMdd')}"
description="string"/>
  </globalVariables>

  <job name="Seoul_Rain" description="서울시 공공 데이터 - 강수량 데이터 수집 Job">
    <schedule>
        <cronExpression>0 * * * * ?</cronExpression>       → Cron Expression을 입력한다.
    </schedule>
    <policy>
      <ingress>
        <local>
          <sourceDirectory conditionType="antPattern">
              <path>/home/hadoop/input/rain</path>  → 로그 파일의 위치
              <condition>rain_*.txt</condition>  → 파일 패턴
          </sourceDirectory>
          <workingDirectory>/tmp/uploader/work</workingDirectory>
          <completeDirectory>/tmp/uploader/complete</completeDirectory>
          <removeAfterCopy>false</removeAfterCopy>
          <errorDirectory>/tmp/uploader/error</errorDirectory>
        </local>
      </ingress>
      <Egress>
        <hdfs cluster="dev">
          <targetPath>/input</targetPath>          → 최종 HDFS 업로드 경로
          <stagingPath>/stage</stagingPath>       → 최종 경로 처리전 스테이지 경로
        </hdfs>
      </Egress>
    </policy>
  </job>
</flamingo>
```

```
#sh start.sh        → HDFS File Uploader를 실행한다.
...
```

Appendix :: 제4절 Apache Pig 설치 가이드

본 문서는 중소기업과 같이 인력을 보유하지 않은 조직에서 Apache Pig를 도입하기 위한 첫 번째 관문인 설치 작업을 실제 현장의 경험자의 경험과 노하우를 바탕으로 설치할 수 있도록 지침을 제공하는 것을 목적으로 한다.

본 가이드는 다음과 같이 이미 Hadoop Cluster가 구축되어 있으며 기본 운영체제는 CentOS를 사용하고, Hadoop Cluster는 네임노드와 데이터노드로 구성되어 있고 모든 노드는 동일한 시스템 구성과 설정 정보를 가지고 있다고 가정한다.



Apache Pig는 Pig Latin Script을 이용하여 데이터를 처리하는 도구로써 Yahoo!의 경우 80% 이상을 Pig를 통해서 작업한다. Apache Pig는 MapReduce로 동작하도록 해주는 일종의 라이브러리이므로 설치가 매우 간단하고 사용하기 쉬우며 MapReduce 관점에서 스크립트를 작성하므로 MapReduce에 익숙한 개발자에게 데이터 처리에 대한 환경을 제공한다.

Apache Pig를 설치하기 위해서 다음과 같이 쉘 환경에서 커맨드를 실행한다. 단, 현재 커맨드를 실행하는 환경은 Apache Hadoop의 환경 변수와 설정 파일이 모두 설정되어 있는 상태이어야 한다.

```
root@i-743-18413-VM:~ #> wget http://ftp.daum.net/apache/pig/pig-0.10.0/pig-0.10.0.tar.gz
root@i-743-18413-VM:~ #> tar xvfz pig-0.10.0.tar.gz
```

```
root@i-743-18413-VM:~ #> ln -s pig-0.10.0 pig
root@i-743-18413-VM:~ #> pig
2011-12-07 18:07:13,958 [main] INFO  org.apache.pig.Main - Logging error messages to:
/root/pig_1323248833954.log
2011-12-07 18:07:14,168 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine -
Connecting to hadoop file system at: hdfs://172.27.135.97:9000
2011-12-07 18:07:14,427 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine -
Connecting to map-reduce job tracker at: 172.27.135.97:9001
grunt> quit
```

Pig가 정상적으로 설치되면 이제 MovieLens 데이터를 가지고 몇 가지 작업을 할 수 있다. 다음 URL의 MovieLens 데이터를 wget 커맨드를 Namenode에서 다운로드하여 해당 디렉터리로 업로드 하도록 한다.

- http://www.openflamingo.org/download/hands_on_lab/movielens/movies.dat
- http://www.openflamingo.org/download/hands_on_lab/movielens/ratings.dat
- http://www.openflamingo.org/download/hands_on_lab/movielens/users.dat
- http://www.openflamingo.org/download/hands_on_lab/movielens/README

```
→ ROOT 계정의 홈 디렉터리에 3개의 로그 파일을 다운로드한다.
root@i-743-18413-VM:~ #> wget http://www.openflamingo.org/download/hands_on_lab/movielens/movies.dat
root@i-743-18413-VM:~ #> wget http://www.openflamingo.org/download/hands_on_lab/movielens/ratings.dat
root@i-743-18413-VM:~ #> wget http://www.openflamingo.org/download/hands_on_lab/movielens/users.dat

→ 다운로드한 *.dat 확장자를 가진 로그 파일을 HDFS에 업로드한다.
root@i-743-18413-VM:~ #> hadoop fs -mkdir /movielens
root@i-743-18413-VM:~ #> hadoop fs -put *.dat /movielens
root@i-743-18413-VM:~ #> hadoop fs -ls /
Found 2 items
drwxr-xr-x   - root supergroup          0 2011-12-07 17:20 /movielens
drwxr-xr-x   - root supergroup          0 2011-12-07 10:04 /tmp
root@i-743-18413-VM:~ #> pig
2011-12-07 18:07:13,958 [main] INFO  org.apache.pig.Main - Logging error messages to:
/root/pig_1323248833954.log
2011-12-07 18:07:14,168 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine -
Connecting to hadoop file system at: hdfs://172.27.135.97:9000
```

```
2011-12-07 18:07:14,427 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to map-reduce job tracker at: 172.27.135.97:9001
```

MovieLens의 movies.dat 파일에서 원하는 원하는 ID, Title만 추출해보도록 하자.

```
root@i-743-18413-VM:~ #> pig
2011-12-07 18:07:13,958 [main] INFO  org.apache.pig.Main - Logging error messages to: /root/pig_1323248833954.log
2011-12-07 18:07:14,168 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: hdfs://172.27.135.97:9000
2011-12-07 18:07:14,427 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to map-reduce job tracker at: 172.27.135.97:9001
grunt> A = LOAD '/movielens/movies.dat' USING PigStorage('^');
grunt> B = FOREACH A GENERATE $0,$1;
grunt> STORE B INTO '/pig_output' USING PigStorage('^');
...
2011-12-07 19:36:43,512 [main] INFO
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
grunt> quit
root@i-743-18413-VM:~/dataset/ml-10M100K #> hadoop fs -ls /
Found 3 items
drwxr-xr-x   - root supergroup          0 2011-12-07 19:32 /movielens
drwxr-xr-x   - root supergroup          0 2011-12-07 19:36 /pig_output
drwxr-xr-x   - root supergroup          0 2011-12-07 19:37 /tmp
root@i-743-18413-VM:~/dataset/ml-10M100K #> hadoop fs -ls /pig_output
Found 2 items
drwxr-xr-x   - root supergroup          0 2011-12-07 19:36 /pig_output/_logs
-rw-r--r--   1 root supergroup     343220 2011-12-07 19:36 /pig_output/part-m-00000
root@i-743-18413-VM:~/dataset/ml-10M100K #> hadoop fs -cat /pig_output/part-m-00000 | head
1^Toy Story (1995)
2^Jumanji (1995)
3^Grumpier Old Men (1995)
4^Waiting to Exhale (1995)
5^Father of the Bride Part II (1995)
6^Heat (1995)
7^Sabrina (1995)
8^Tom and Huck (1995)
9^Sudden Death (1995)
10^GoldenEye (1995)
```

Appendix :: 제5절 MapReduce 프로그래밍 템플릿

본 문서는 Big Data 프로젝트를 수행할 때 Hadoop MapReduce 프로그램을 작성해야 하는 경우 사용할 수 있는 프로그래밍 템플릿이다.

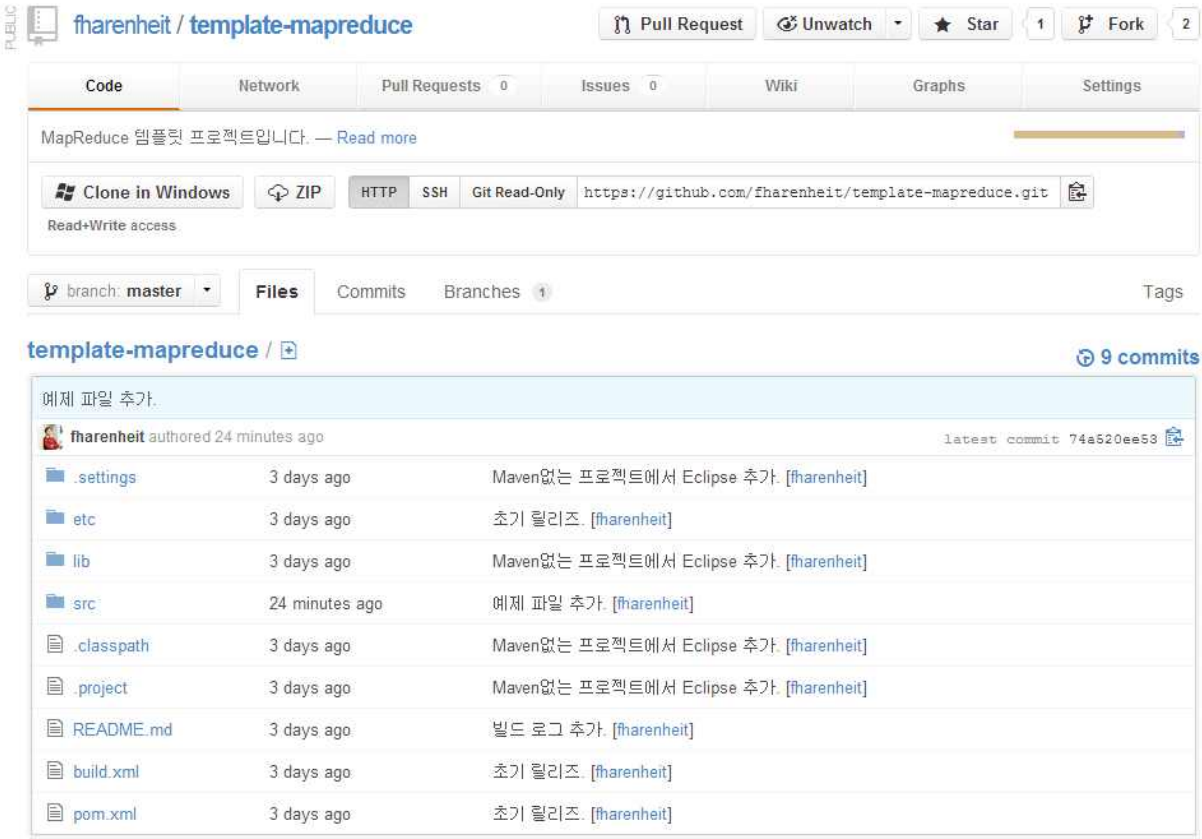
1. 템플릿 구성 요소

MapReduce 프로그래밍 템플릿은 다음을 지원하기 위해서 제공한다.

- ○ Apache Hadoop의 MapReduce 개발 환경
- ○ Apache Pig, Hive의 Function 개발 환경
- ○ 품질 향상을 위한 MapReduce 단위 테스트 환경

2. 템플릿 다운로드

MapReduce 프로그래밍 템플릿은 https://github.com/fharenheit/template-mapreduce 에서 다운로드할 수 있다.



3. 환경 요구사항

본 프로젝트 템플릿은 다음의 환경을 기반으로 구성되었다.

○ JDK 1.6 이상 (JDK 1.6 권장)

□ JDK 1.6 Download :

http://www.oracle.com/technetwork/java/javase/downloads

○ Apache Maven 3.x 이상

□ Windows :

http://ftp.daum.net/apache/maven/maven-3/3.0.4/binaries/apache-maven-3.0.4-bin.zip

□ Linux, MacOSX :

http://ftp.daum.net/apache/maven/maven-3/3.0.4/binaries/apache-maven-3.0.4-bin.tar.gz

○ Apache Ant 1.8 이상

□ Windows : http://ftp.daum.net/apache/ant/binaries/apache-ant-1.8.4-bin.zip

□ Linux, MacOSX :

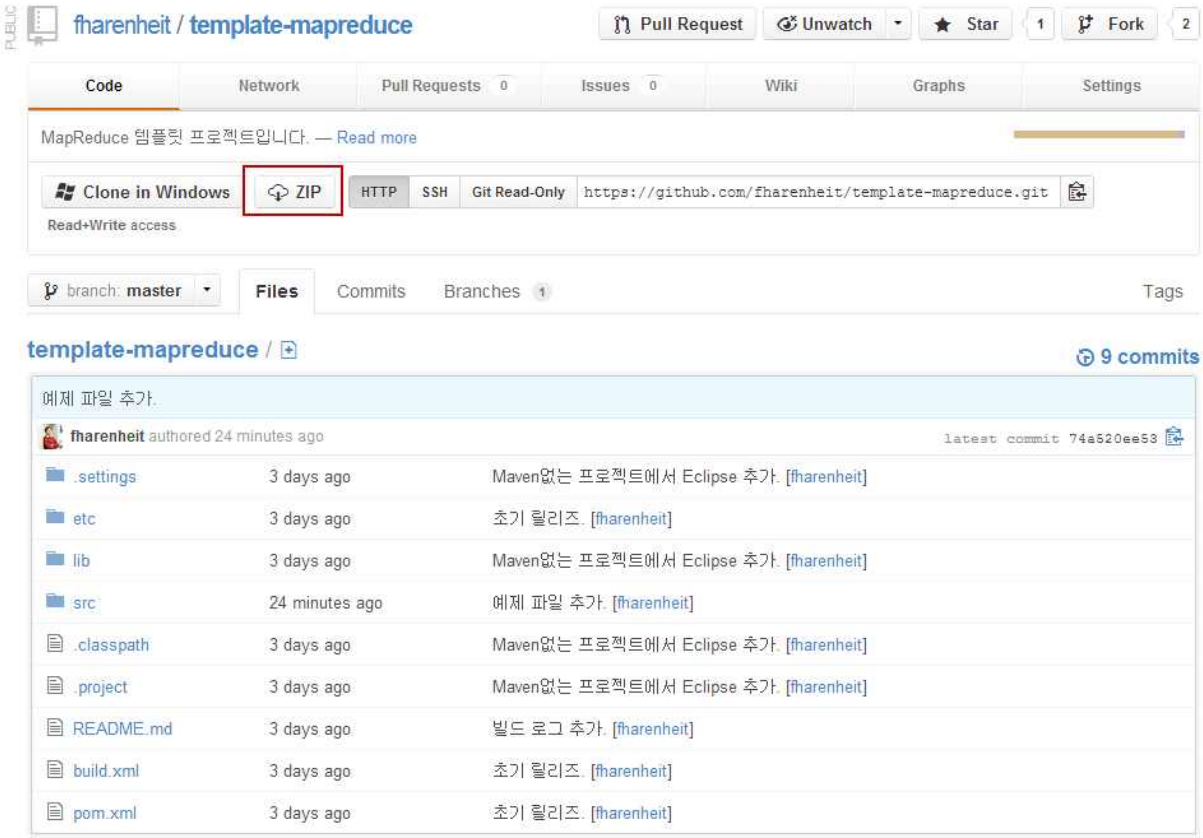
http://ftp.daum.net/apache/ant/binaries/apache-ant-1.8.4-bin.tar.gz

○ Eclipse (M2Eclipse 플러그인 설치 필요) 또는 IntelliJ IDEA

→ M2Eclipse Update Site : http://download.eclipse.org/technology/m2e/releases

4. 템플릿 다운로드

프로젝트 템플릿을 다운로드 하려면 다음의 zip 링크를 클릭한다.



5. 소스코드 빌드 및 MapReduce Job JAR 파일 패키징

다음의 커맨드를 실행하면 MapReduce Job JAR 파일을 패키징할 수 있다.

```
#mvn package
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Flamingo MapReduce Template 0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.4.1:clean (default-clean) @ mapreduce-template ---
```

```
[INFO] Deleting C:\Users\Cloudine\Desktop\mapreduce-template-0.1\target
[INFO]
[INFO] --- maven-resources-plugin:2.5:resources (default-resources) @ mapreduce-template ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) @ mapreduce-template ---
[INFO] Compiling 40 source files to C:\Users\Cloudine\Desktop\mapreduce-template-0.1\target\classes
[INFO]
[INFO] --- maven-resources-plugin:2.5:testResources (default-testResources) @ mapreduce-template ---
[debug] execute contextualize
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:testCompile (default-testCompile) @ mapreduce-template ---
[INFO] Compiling 1 source file to C:\Users\Cloudine\Desktop\mapreduce-template-0.1\target\test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.10:test (default-test) @ mapreduce-template ---
[INFO] Surefire report directory: C:\Users\Cloudine\Desktop\mapreduce-template-0.1\target\surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running org.openflamingo.mapreduce.etl.groupby.GroupByMapReduceTest
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.6 sec

Results :

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-dependency-plugin:2.1:unpack (unpack) @ mapreduce-template ---
[INFO] Configured Artifact: com.google.guava:guava:r09:jar
[INFO] Configured Artifact: org.apache.mahout.commons:commons-cli:2.0-mahout:jar
[INFO] Configured Artifact: commons-cli:commons-cli:1.2:jar
[INFO] Configured Artifact: commons-lang:commons-lang:2.5:jar
[INFO] Unpacking C:\Users\Cloudine\.m2\repository\com\google\guava\guava\r09\guava-r09.jar to
  C:\Users\Cloudine\Desktop\mapreduce-template-0.1\target\classes
   with includes null and excludes:null
[INFO] Unpacking
C:\Users\Cloudine\.m2\repository\org\apache\mahout\commons\commons-cli\2.0-mahout\commons-cli-2.0-mahout.ja
r to
  C:\Users\Cloudine\Desktop\mapreduce-template-0.1\target\classes
   with includes null and excludes:null
[INFO] Unpacking C:\Users\Cloudine\.m2\repository\commons-cli\commons-cli\1.2\commons-cli-1.2.jar to
  C:\Users\Cloudine\Desktop\mapreduce-template-0.1\target\classes
```

```
   with includes null and excludes:null
[INFO] Unpacking C:\Users\Cloudine\.m2\repository\commons-lang\commons-lang\2.5\commons-lang-2.5.jar to
  C:\Users\Cloudine\Desktop\mapreduce-template-0.1\target\classes
   with includes null and excludes:null
[INFO]
[INFO] --- maven-jar-plugin:2.3.2:jar (default-jar) @ mapreduce-template ---
[INFO] Building jar:
C:\Users\Cloudine\Desktop\mapreduce-template-0.1\target\mapreduce-template-0.1-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 9.110s
[INFO] Finished at: Sun Feb 03 03:41:10 KST 2013
[INFO] Final Memory: 21M/328M
[INFO] ------------------------------------------------------------------------
```

6. MapReduce 프로그래밍

MapReduce Job을 최소의 노력으로 Hadoop에서 실행하려면 다음의 클래스를 작성해야 한다.

- ○ MapReduce Job의 파라미터 처리, 설정 정보 구성 및 MapReduce Job을 실행하는 Driver
- ○ 처리할 입력 로그 경로의 파일을 로딩하여 Key Value로 출력하는 Mapper
- ○ Mapper의 출력 Key Value로 취합하여 처리하는 Reducer
- ○ 그 외 기타 Combiner, Partitioner, DistributedCache 등등

6.1 Driver 작성하기

Driver는 다음과 같이 Configured, Tool을 상속 및 구현하며 최소 형식은 다음과 같다.

```
public class SampleDriver extends org.apache.hadoop.conf.Configured
                          implements org.apache.hadoop.util.Tool {

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new SampleDriver(), args);
        System.exit(res);
    }
```

```
    public int run(String[] args) throws Exception {
        Job job = new Job();

        ...

        job.setJarByClass(SampleDriver.class);

        // Mapper Class
        job.setMapperClass(SampleMapper.class);

        // Output Key/Value
        job.setMapOutputKeyClass(NullWritable.class);
        job.setMapOutputValueClass(Text.class);

        // Reducer Task
        job.setNumReduceTasks(0);

        // Run a Hadoop Job
        return job.waitForCompletion(true) ? 0 : 1;
    }

}
```

6.2. 커맨드 라인 파라미터 처리하기

커맨드 라인을 처리하는 작업은 매우 중요하며 본 프로젝트 템플릿에서는 다음의 두 가지 경우를 제공한다.

○ 단순 파라미터 처리

□ 장점 : 코드가 단순

□ 단점 : 필수 파라미터 검증을 직접 구성해야 함, 파라미터가 Linux 표준형식이 아님

○ Linux 형식의 파라미터 처리

□ 장점 : Linux 형식의 파라미터(긴 이름, 짧은 이름) 지원, 필수 파라미터 검증

□ 단점 : 상대적으로 코드 복잡도 증가

6.2.1 형식 1

이 형식은 가장 단순하게 구현할 수 있는 방법으로 parseArguements() 메소드에서 파라미터

를 처리한다. 항상 파라미터명 다음에 파라미터의 값을 지정해야 한다. 다음 예제는 org.openflamingo.mapreduce.sample.SampleDriver을 참고하도록 한다.

```
public class SampleDriver extends org.apache.hadoop.conf.Configured
                          implements org.apache.hadoop.util.Tool {

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new SampleDriver(), args);
        System.exit(res);
    }

    public int run(String[] args) throws Exception {
        Job job = new Job();
        parseArguements(args, job);

        job.setJarByClass(SampleDriver.class);

        // Mapper Class
        job.setMapperClass(SampleMapper.class);

        // Output Key/Value
        job.setMapOutputKeyClass(NullWritable.class);
        job.setMapOutputValueClass(Text.class);

        // Reducer Task
        job.setNumReduceTasks(0);

        // Run a Hadoop Job
        return job.waitForCompletion(true) ? 0 : 1;
    }

    private void parseArguements(String[] args, Job job) throws IOException {
        for (int i = 0; i < args.length; ++i) {
            if ("-input".equals(args[i])) {
                FileInputFormat.addInputPaths(job, args[++i]);
            } else if ("-output".equals(args[i])) {
                FileOutputFormat.setOutputPath(job, new Path(args[++i]));
            } else if ("-jobName".equals(args[i])) {
                job.getConfiguration().set("mapred.job.name", args[++i]);
            } else if ("-columnToClean".equals(args[i])) {
                job.getConfiguration().set("columnToClean", args[++i]);
            } else if ("-delimiter".equals(args[i])) {
                job.getConfiguration().set("delimiter", args[++i]);
            }
        }
    }
```

```
}
```

위 형식을 실행하려면 다음과 같이 커맨드를 실행한다.

```
#hadoop jar <JAR_FILE> org.openflamingo.mapreduce.sample.SampleDriver --input <IN> --output <OUT> ..
```

6.2.2 형식 2

이 형식은 복잡하지만 도움말과 필수 옵션을 처리할 수 있는 기능을 사용할 수 있다. 다음 예제는 org.openflamingo.mapreduce.sample.Sample2Driver을 참고 하도록 한다.

```
public class Sample2Driver extends org.apache.hadoop.conf.Configured
                           implements org.apache.hadoop.util.Tool {

    /**
     * 필수 옵션
     */
    private final String[][] requiredOptions =
            {
                    {"i", "입력 경로를 지정해 주십시오. 입력 경로가 존재하지 않으면 MapReduce가 동작할 수
없습니다."},
                    {"o", "출력 경로를 지정해 주십시오."},
                    {"d", "컬럼의 구분자를 지정해주십시오. CSV 파일의 컬럼을 처리할 수 없습니다."},
            };

    /**
     * 사용가능한 옵션 목록을 구성한다.
     *
     * @return 옵션 목록
     */
    private static Options getOptions() {
        Options options = new Options();
        options.addOption("i", "input", true, "입력 경로 (필수)");
        options.addOption("o", "output", true, "출력 경로 (필수)");
        options.addOption("d", "delimiter", true, "컬럼 구분자 (필수)");
        options.addOption("od", "delete", false, "출력 경로가 이미 존재하는 경우 삭제");
        return options;
    }

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Sample2Driver(), args);
        System.exit(res);
    }
```

```
    public int run(String[] args) throws Exception {
        Job job = new Job();

        int result = parseArguements(args, job);
        if (result != 0) {
            return result;
        }

        job.setJarByClass(Sample2Driver.class);

        // Mapper Class
        job.setMapperClass(SampleMapper.class);

        // Output Key/Value
        job.setMapOutputKeyClass(NullWritable.class);
        job.setMapOutputValueClass(Text.class);

        // Reducer Task
        job.setNumReduceTasks(0);

        // Run a Hadoop Job
        return job.waitForCompletion(true) ? 0 : 1;
    }

    private int parseArguements(String[] args, Job job) throws Exception {
        ////////////////////////////////////////
        // 옵션 목록을 구성하고 검증한다.
        ////////////////////////////////////////

        Options options = getOptions();
        HelpFormatter formatter = new HelpFormatter();
        if (args.length == 0) {
            formatter.printHelp("org.openflamingo.hadoop jar <JAR> " + getClass().getName(), options,
true);
            return -1;
        }

        // 커맨드 라인을 파싱한다.
        CommandLineParser parser = new BasicParser();
        CommandLine cmd = parser.parse(options, args);

        // 파라미터를 검증한다.
        for (String[] requiredOption : requiredOptions) {
            if (!cmd.hasOption(requiredOption[0])) {
                formatter.printHelp("org.openflamingo.hadoop jar <JAR> " + getClass().getName(), options,
true);
```

```
                return -1;
            }
        }

        ////////////////////////////////////
        // 파라미터를 Hadoop Configuration에 추가한다
        ////////////////////////////////////

        if (cmd.hasOption("i")) {
            FileInputFormat.addInputPaths(job, cmd.getOptionValue("i"));
        }

        if (cmd.hasOption("o")) {
            FileOutputFormat.setOutputPath(job, new Path(cmd.getOptionValue("o")));
        }

        if (cmd.hasOption("d")) {
            job.getConfiguration().set("delimiter", cmd.getOptionValue("d"));
        }

        // 옵션을 지정한 경우 출력 경로를 삭제한다.
        if (cmd.hasOption("od")) {
            if (HdfsUtils.isExist(cmd.getOptionValue("o"))) {
                HdfsUtils.deleteFromHdfs(cmd.getOptionValue("o"));
            }
        }

        return 0;
    }
}
```

위 형식을 실행하려면 다음과 같이 커맨드를 실행한다.

```
#hadoop jar <JAR_FILE> org.openflamingo.mapreduce.sample.SampleDriver --input <IN> --output <OUT> ..
```

7. 커맨드 라인 파라미터에 JVM Args 넘기기

종종 MapReduce Job의 Configuration에 자동으로 값을 설정해야 하는 경우 형식2의 파라미터 처리를 이용하고 다음과 같이 커맨드 라인을 입력한다.

```
#hadoop jar <JAR_FILE> org.openflamingo.mapreduce.sample.SampleDriver \
                    -Dmapred.job.name="Test MapReduce Job" --input <IN> --output <OUT> ..
```

위 커맨드 라인 예제에서 -Dmapred.job.name="Test MapReduce Job"와 같이 파라미터를 추가하면 Job의 Configuration에 자동으로 추가된다. 단, -D 커맨드는 반드시 클래스명 뒤에서만 사용해야 한다.

8. Mapper 작성하기

Mapper를 작성할 때에는 Mapper 클래스를 상속하고 입출력 파라미터의 형식을 다음과 같이 Generic으로 정의한다. 그리고 난 후 Eclipse 또는 IntelliJ IDEA에서 method override 기능을 이용하여 cleanup, map, setup 메소드를 오버라이드 하도록 한다. 이 때 정상적으로 오버라이드가 되었다면 @Override annotation을 추가했을 때 아무런 문제가 없어야 하며 @Override annotation은 반드시 추가하도록 한다.

```
public class WordcountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    private String delimiter;

    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
        Configuration configuration = context.getConfiguration();
        delimiter = configuration.get("delimiter");
    }

    @Override
    protected void map(LongWritable key, Text value, Context context)
                                                    throws IOException, InterruptedException {
        String row = value.toString();
        String[] columns = row.split(delimiter);
        for (String word : columns) {
            context.write(new Text(word), new IntWritable(1));
        }
    }

    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
    }
}
```

9. Reducer 작성하기

Reducer를 작성할 때에는 Reducer 클래스를 상속하고 입출력 파라미터의 형식을 다음과 같

이 Generic으로 정의한다. 그리고 난 후 Eclipse 또는 IntelliJ IDEA에서 method override 기능을 이용하여 cleanup, map, setup 메소드를 오버라이드 하도록 한다. 이 때 정상적으로 오버라이드가 되었다면 @Override annotation을 추가했을 때 아무런 문제가 없어야 하며 @Override annotation은 반드시 추가하도록 한다.

```
public class WordcountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
    }

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context)
                                                    throws IOException, InterruptedException {
        Iterator<IntWritable> iterator = values.iterator();
        int sum = 0;
        while (iterator.hasNext()) {
            IntWritable one = iterator.next();
            sum += one.get();
        }
        context.write(key, new IntWritable(sum));
    }

    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
    }
}
```

10. Program Driver에 MapReduce Driver 등록하기

MapReduce Job을 실행하는 Driver를 실행하려면 패키지명을 포함한 Driver 클래스명을 입력해야 한다.

```
public class MapReduceDriver {

    public static void main(String argv[]) {
        ProgramDriver programDriver = new ProgramDriver();
        try {
            programDriver.addClass("gropuby", GroupByDriver.class, "Group By MapReduce Job");
            programDriver.driver(argv);

            System.exit(Constants.JOB_SUCCESS);
        } catch (Throwable e) {
```

```
            e.printStackTrace();
            System.exit(Constants.JOB_FAIL);
        }
    }
}
```

이렇게 등록한 MapReduce Job Driver는 다음과 같이 alias로 실행할 수 있다.

```
#hadoop jar <JAR_FILE> groupby  ... --input <IN> --output <OUT> ..
```

11. MapReduce Job JAR 파일에 다른 라이브러리를 같이 패키징하는 방법

Apache Maven의 POM 파일은 pom.xml 파일에 다음과 같이 maven-dependency-plugin 플러그인이 정의되어 있다. 이 플러그인에 MapReduce Job JAR 파일 내에 외부 라이브러리를 같이 패키징하고자 하는 Maven Artifact를 artifactItem 항목으로 추가하도록 한다. 이런 방식의 패키징을 해야 하는 이유는 Hadoop을 설치한 이후에는 Hadoop은 시스템을 관리하는 관리자의 허가를 통해서 수정해야 하나 MapReduce는 개발자가 관리하기 때문이다.

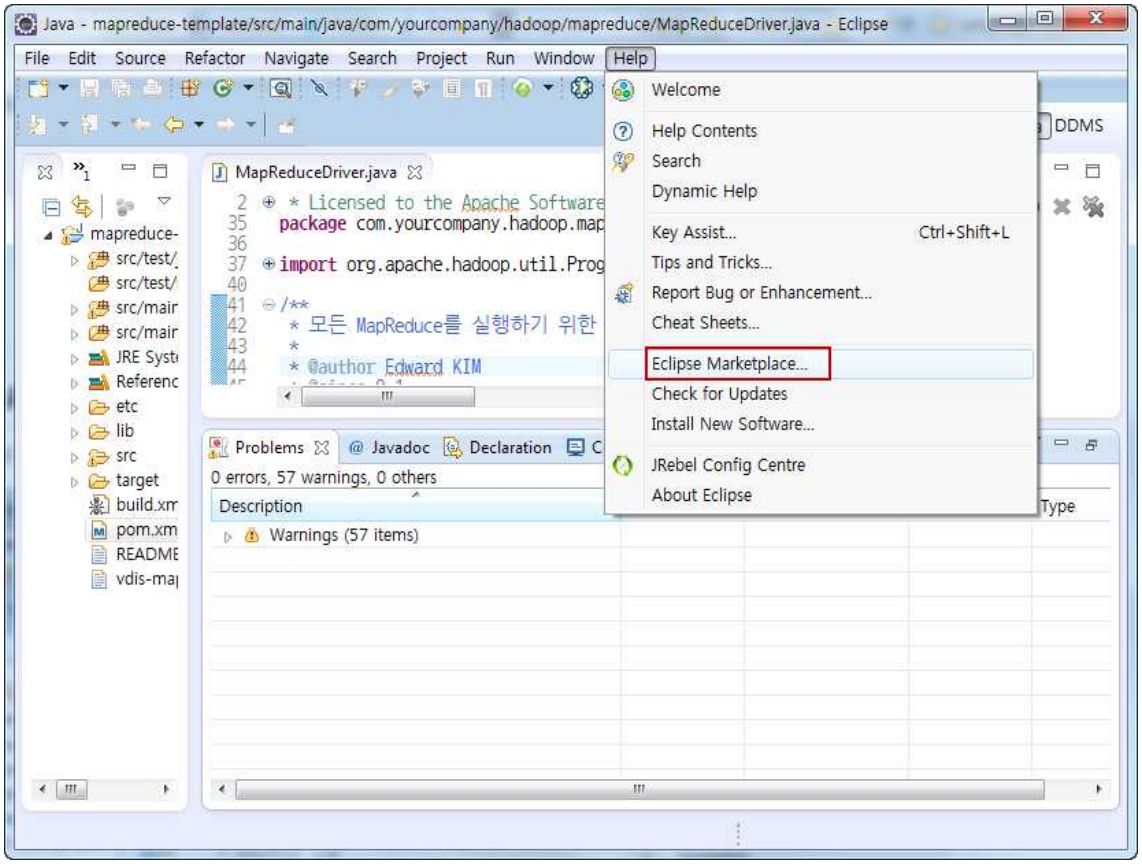
```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-dependency-plugin</artifactId>
    <executions>
        <execution>
            <id>unpack</id>
            <phase>test</phase>
            <goals>
                <goal>unpack</goal>
            </goals>
            <configuration>
                <outputDirectory>${project.build.outputDirectory}</outputDirectory>
                <artifactItems>
                    <artifactItem>
                        <groupId>com.google.guava</groupId>
                        <artifactId>guava</artifactId>
                        <version>${guava.version}</version>
                    </artifactItem>
                    <artifactItem>
                        <groupId>org.apache.mahout.commons</groupId>
                        <artifactId>commons-cli</artifactId>
                        <version>${commons.cli2.version}</version>
                    </artifactItem>
                    <artifactItem>
```

```
                    <groupId>commons-cli</groupId>
                    <artifactId>commons-cli</artifactId>
                    <version>${commons.cli.version}</version>
                </artifactItem>
                <artifactItem>
                    <groupId>commons-lang</groupId>
                    <artifactId>commons-lang</artifactId>
                    <version>${commons.lang.version}</version>
                </artifactItem>
            </artifactItems>
        </configuration>
    </execution>
</executions>
</plugin>
```

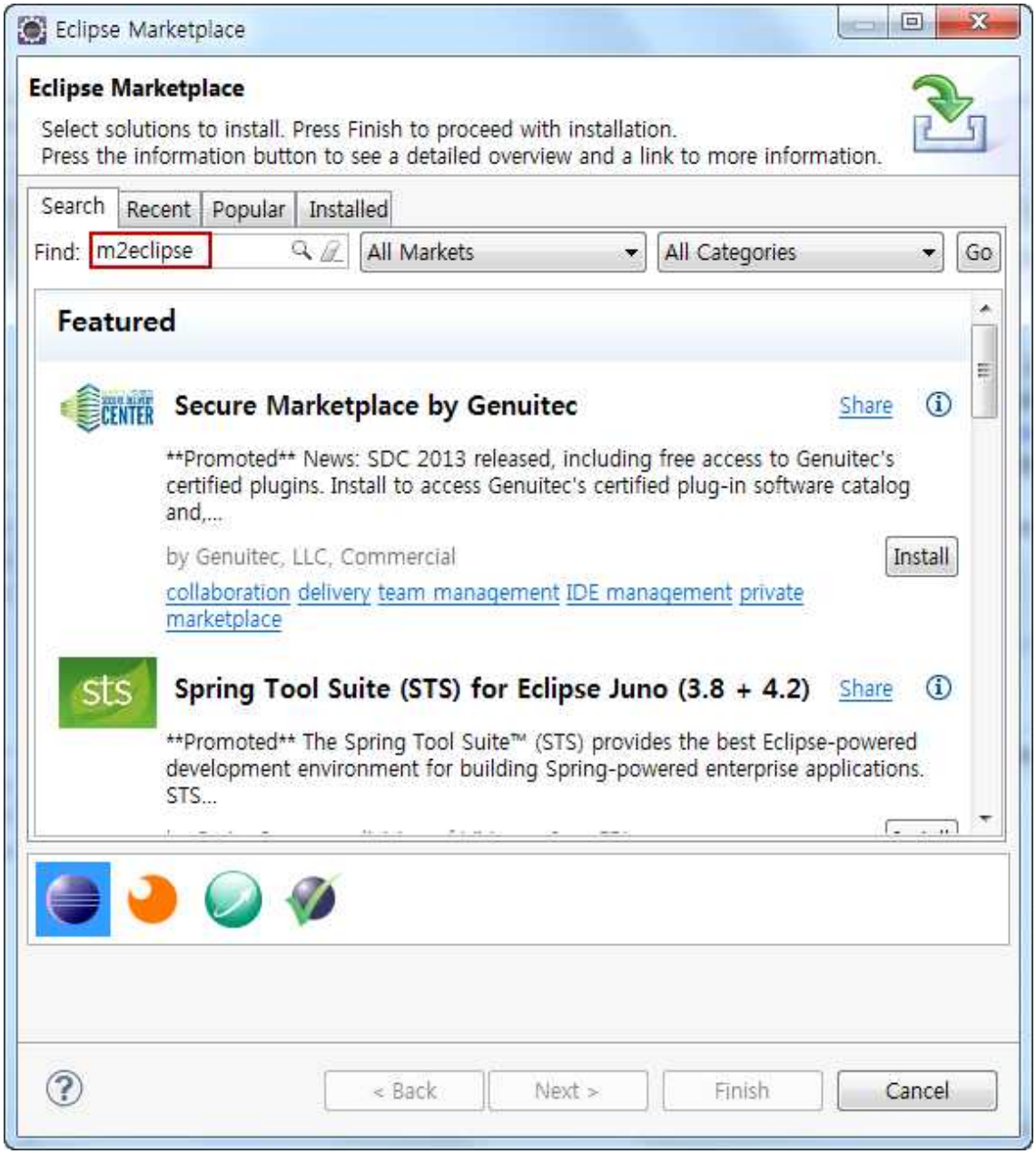
이렇게 등록한 MapReduce Job Driver는 다음과 같이 alias로 실행할 수 있다.

12. M2Eclipse 플러그인 설치

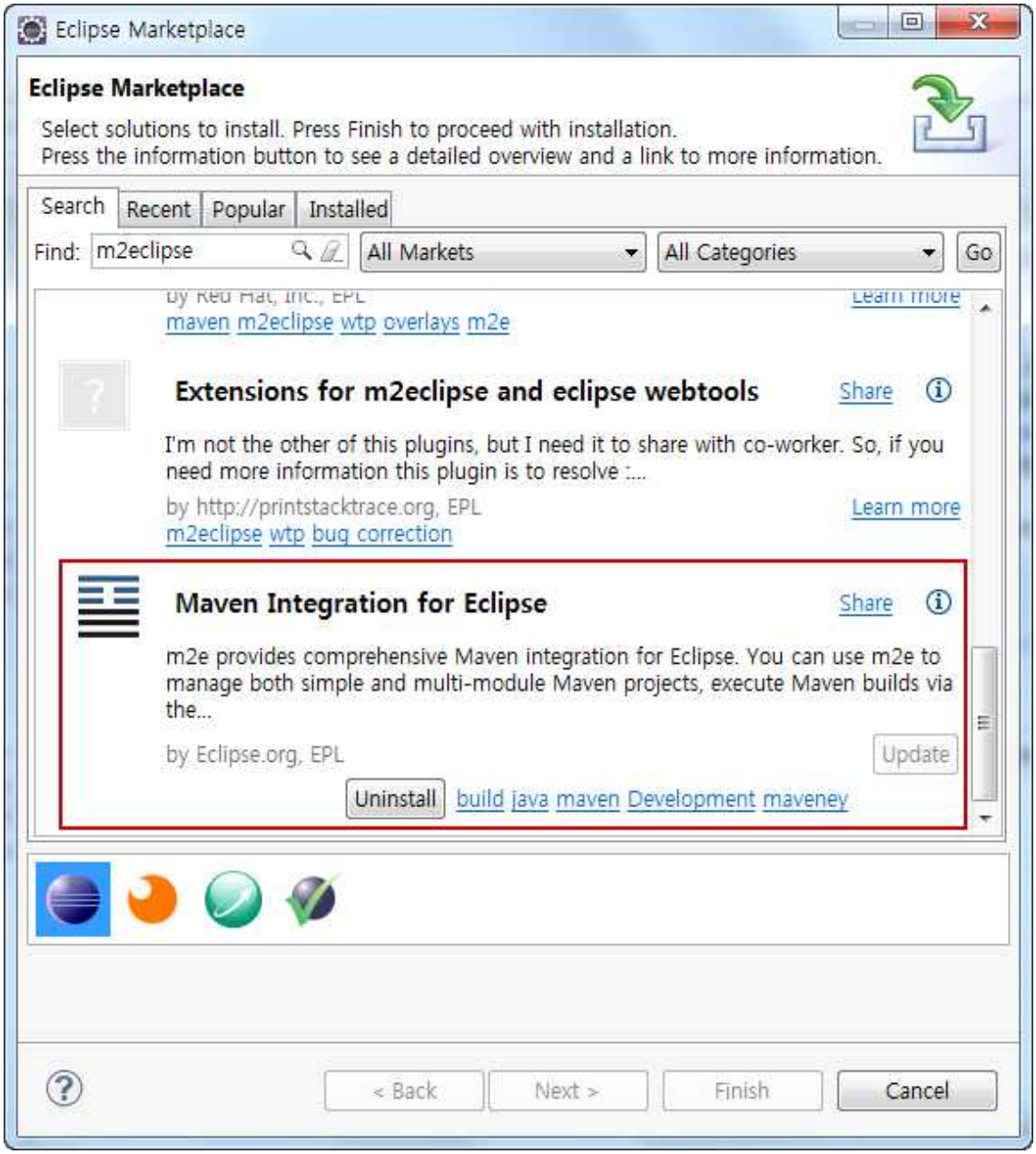
Eclipse에서 MapReduce Job을 Hadoop Cluster에서 실행하기 위해서 Maven으로 빌드 및 패키징을 해야 한다. 이를 위해서 다음의 순서대로 진행하도록 한다. 우선 다음과 같이 Eclipse 메뉴에서 Eclipse Marketplace를 선택한다.



그리고 나서 다음의 창이 나타나면 "m2eclipse"를 입력하고 엔터를 누른다.

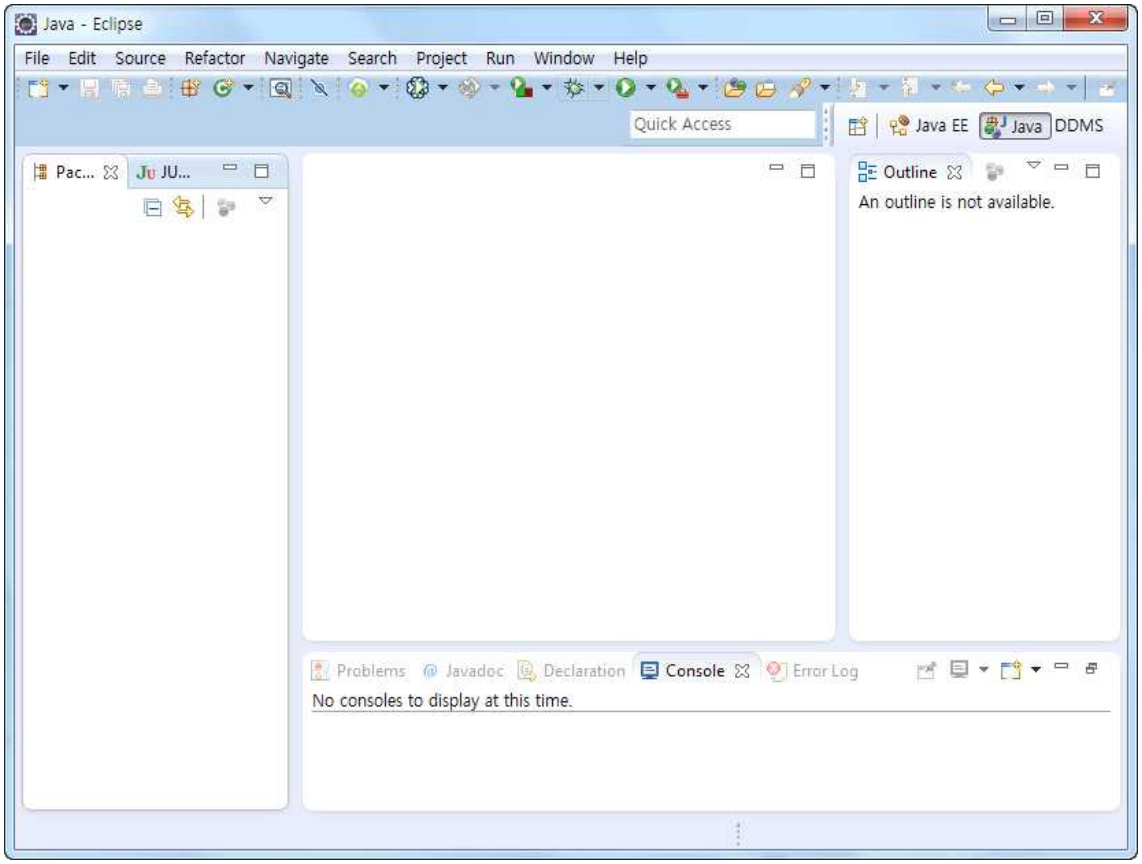


관련 플러그인이 나타나면 다음을 찾아서 설치한다.

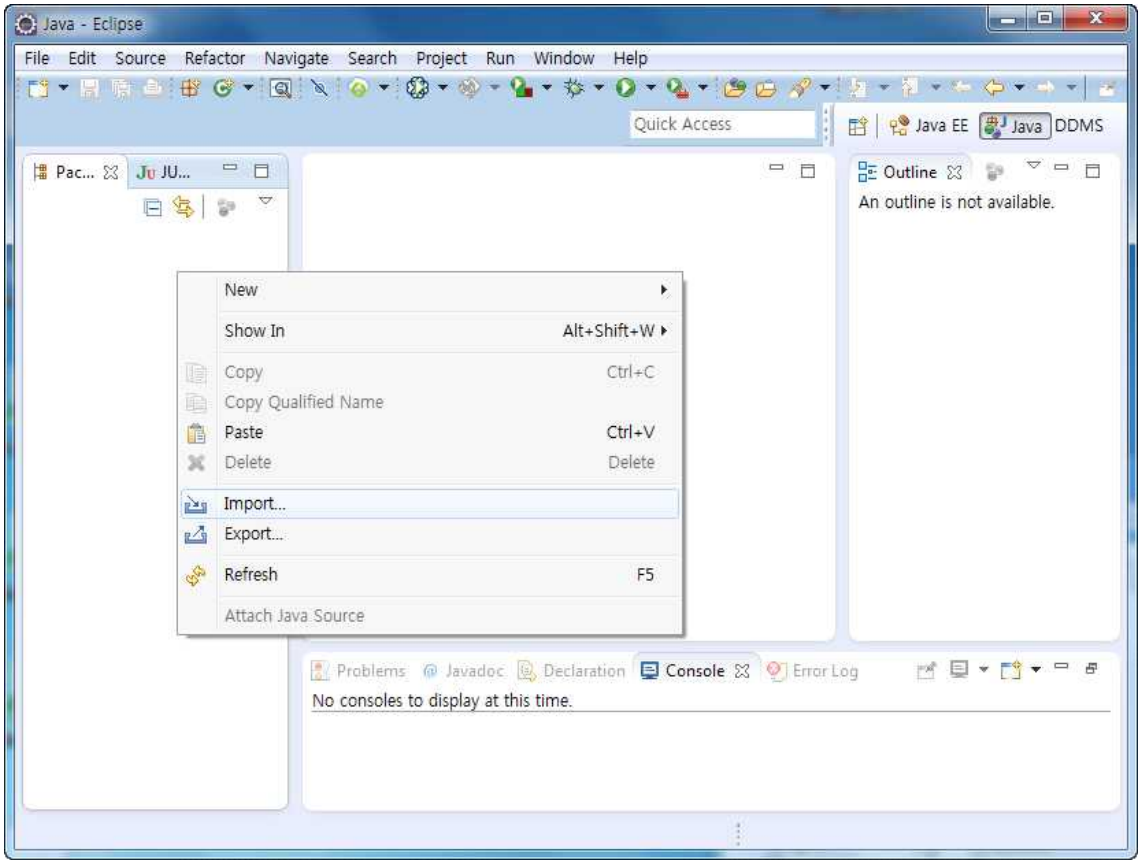


13. Eclipse에서 프로젝트 템플릿 import하기

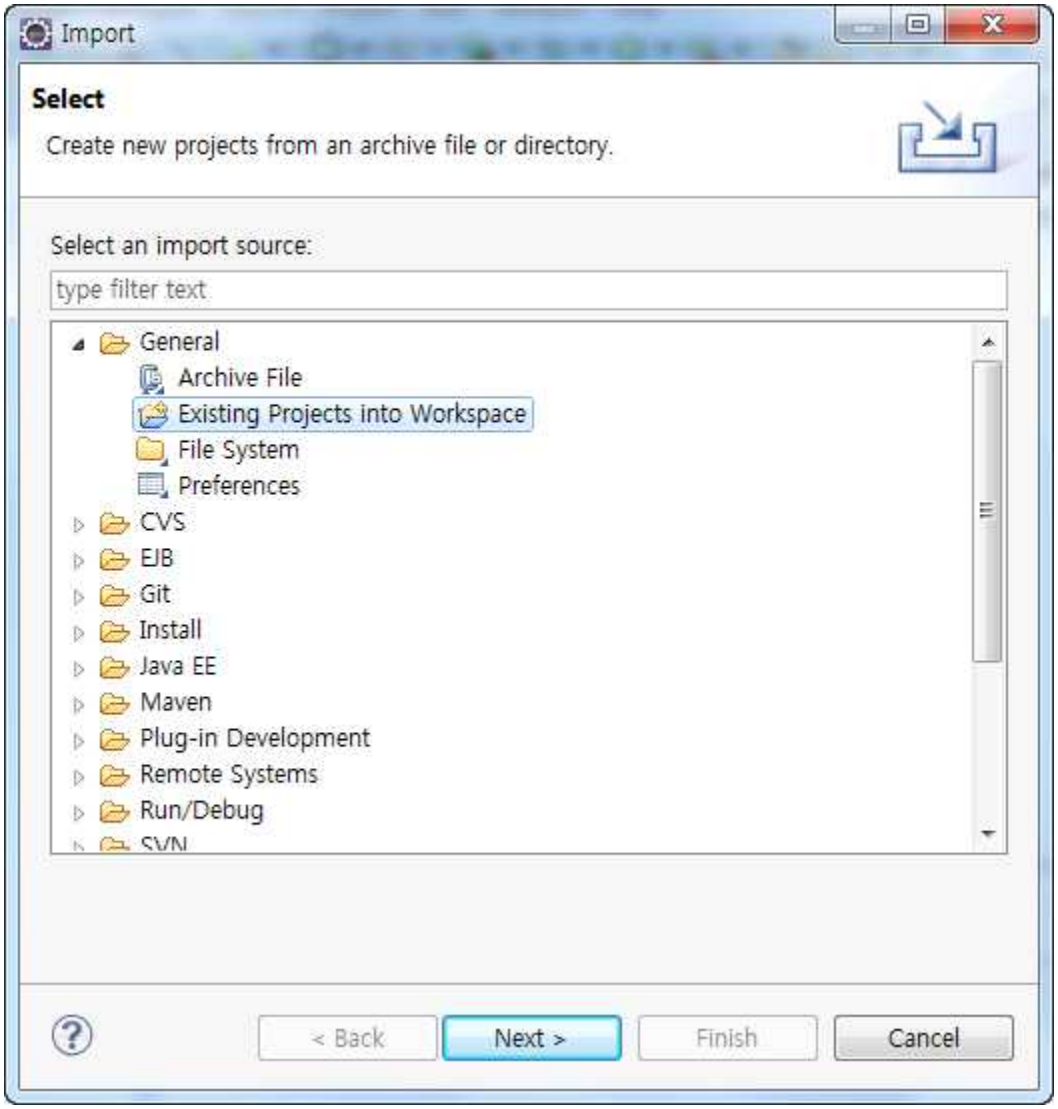
Eclipse에서 프로젝트를 Import하기 위해서Eclipse Project로 다음의 순서에 따라서 import할 수 있다. 우선 Eclipse를 다음과 같이 실행한다.



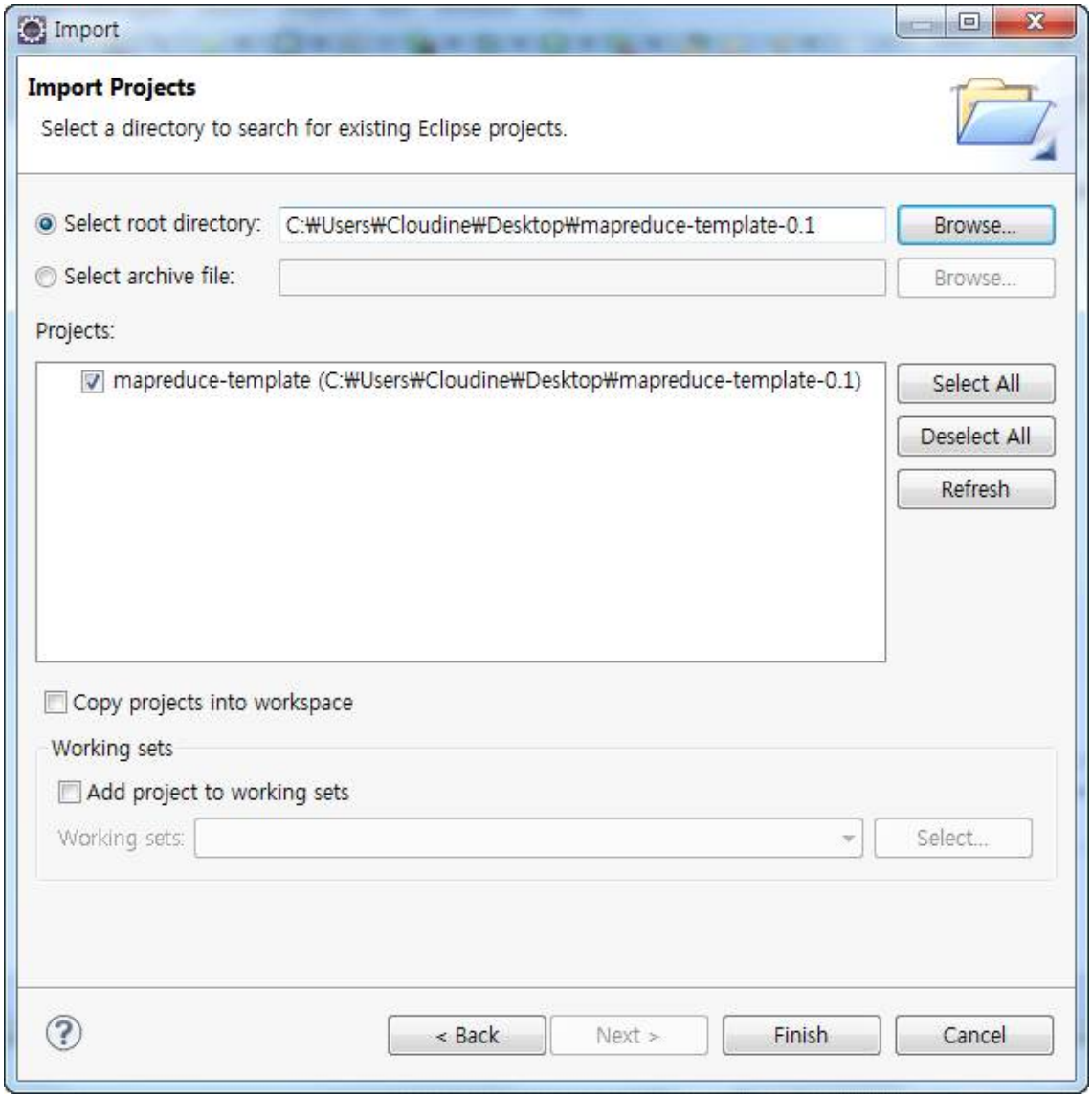
이제 다음과 같이 오른쪽 버튼을 선택하여 import를 선택한다.

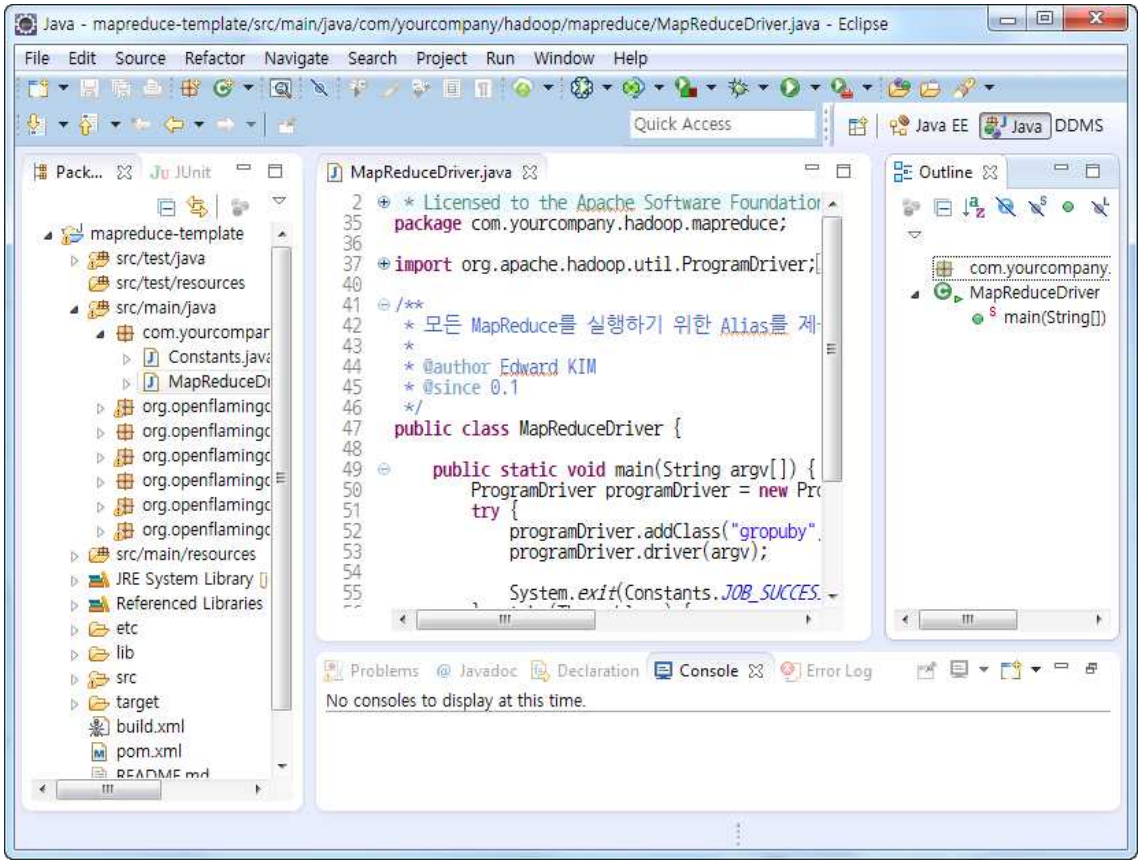


Eclipse 프로젝트를 import하기 위해서 Existing Projects into Workspace를 선택한다.



Github에서 다운로드한 소스코드을 압축해제한 후 그 경로를 다음과 같이 지정한다.

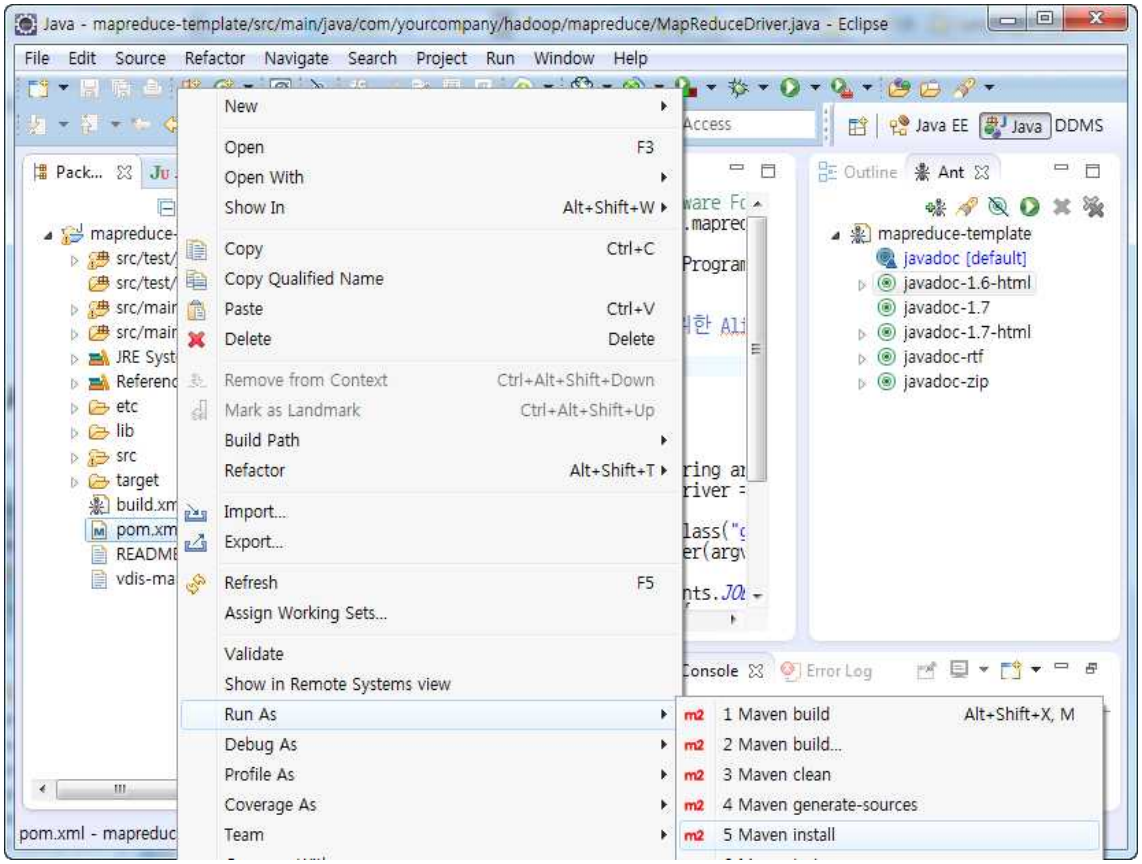

Import
Import Projects
Select a directory to search for existing Eclipse projects.
Select root directory:
C:₩Users₩Cloudine₩Desktop₩mapreduce-template-0.1
Browse...
Select archive file:
Browse...
Projects:
mapreduce-template (C:₩Users₩Cloudine₩Desktop₩mapreduce-template-0.1)
Select All
Deselect All
Refresh
Copy projects into workspace
Working sets
Add project to working sets
Working sets:
Select...
< Back
Next >
Finish
Cancel



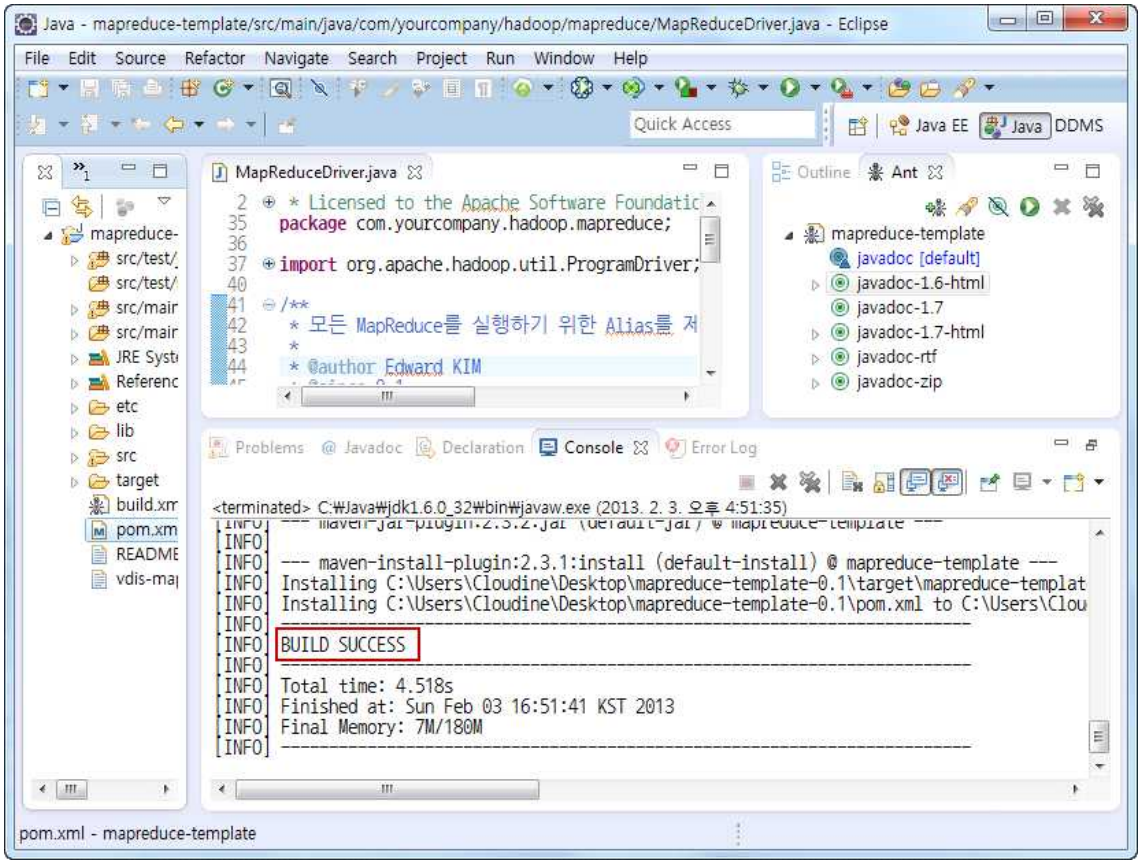
정상적으로 import가 되면 다음과 같이 소스코드가 이상 없이 편집가능한 상태임을 확인할 수 있다.

14. Eclipse에서 빌드하기

Eclipse에서 빌드하기 위해서 M2Eclipse 플러그인을 설치하고 다음과 같이 pom.xml 파일에서 마우스 오른쪽 버튼을 누르고 나서 maven install 또는 maven package 과정을 실행한다.



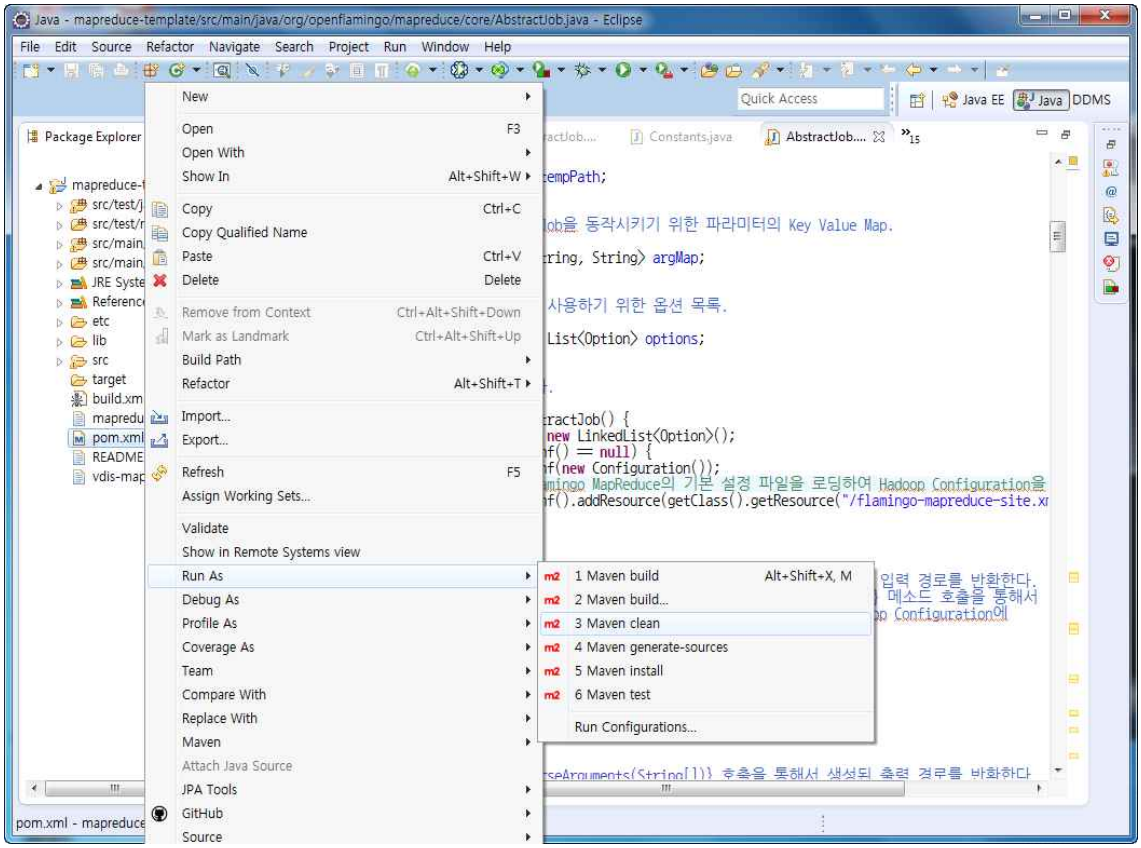
〈PROJECT_HOME〉/target/mapreduce-template-0.1-SNAPSHOT.jar 파일이 생성된다.



이렇게 생성된 파일을 Hadoop을 실행할 수 있는 서버로 FTP 업로드후 다음의 커맨드를 실행한다.

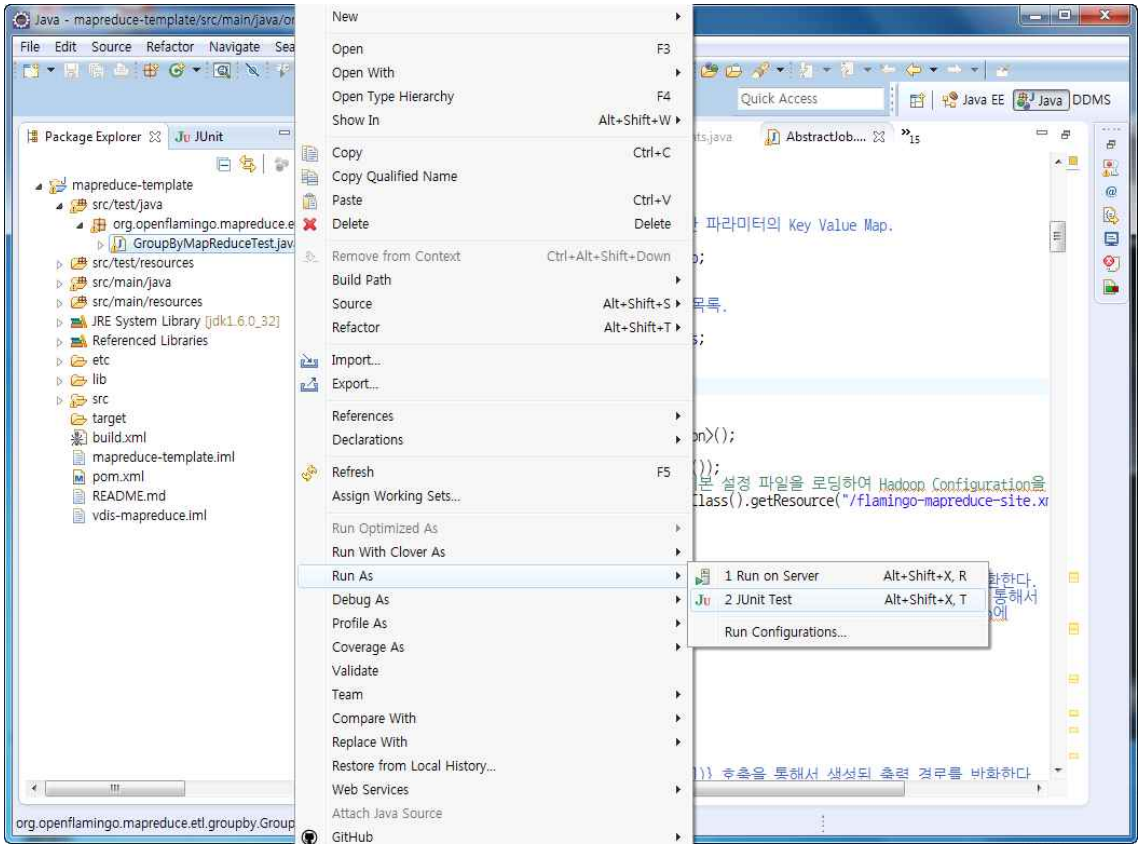
```
#hadoop jar mapreduce-template-0.1-SNAPSHOT.jar groupby  ... --input <IN> --output <OUT> ..
```

종종 프로젝트 소스코드를 변경하여 maven install 또는 maven package를 하는 경우 제대로 반영되지 않는 상황이 발생하곤 하는데 이 경우 maven clean 작업을 다음과 같이 먼저 실행하고 maven package 또는 maven install을 하도록 한다.



15. 단위 테스트 하기

MRUnit을 이용하여 작성한 MapReduce를 테스트하기 위해서는 다음과 같이 MRUnit을 이용하여 작성한 JUnit TestCase를 실행하도록 한다.



MRUnit에 대한 상세한 설명은 “Hadoop 없이 MapReduce 테스트하기[7]“을 참고하도록 한다.

7) https://github.com/fharenheit/template-mapreduce/wiki/MRUnit.pdf

[참고문헌]

- Hadoop In Practice, Manning
- Esper Documentation, Espertech
- Hadoop Operations, Oreilly
- Hadoop The Definitive Guide 3rd, Oreilly
- Apache S4 Documentation, Apache