하둡 기반의 데이터 웨어하우스 구현을 위한 인터랙티브 및 스트림 처리 기술

Interactive Analysis and In-stream Processing for Implmenting Hadoop-Based Data Warehouse

이정룡 · 박근태 · 장홍성
Jungryong Lee · Keuntae Park · Hongsung Chang

기존 Data Warehouse(DW)는 시스템의 확장성이 제한되고 분석의 대상이 요약(Summary)된 데이터로 한정되는 문제를 가지고 있다. 이를 해결하기 위해서 Hadoop으로 대표되는 Big Data 기술을 적용하려는 시도가 활발히 진행되고 있지만 Batch 처리 기반의 Hadoop MapReduce 기술은 DW가 요구하는 데이터의 신속한 처리를 통한 빠른 응답에는 적합하지 않다. 본 논문은 이에 대한 해결책으로 빠른 응답을 위한 Interactive Analysis 기술과 실시간 처리를 위한 In-stream processing 기술을 제시하였고 실험 결과는 위의 기술들이 Hadoop 기반 DW의 구현에 충분한 성능을 제공할 수 있음을 보인다. 향후에는 R, OLAP 등을 활용한 고급 분석 영역으로 Hadoop 기반 기술을 확장하고 사용자 편의를 위해 Interactive Analysis와 In-stream processing 기술을 유기적으로 통합할 계획을 가지고 있다.

주제어: 하둡, 데이터 웨어하우스, 맵리듀스, 인터랙티브 분석, 스트림 처리, 실시간 처리

Data warehouse (DW) has limitation on scalability and also limited scope of analysis as only summarized data is used. There are lots of on-going works to overcome the limitations by applying big data technologies represented by Hadoop. However, they induce another problem of high latency from the MapReduce, which is originally designed for batch processing. To solve that problem, this paper suggests interactive analysis for fast response and in-stream processing for real-time processing, and experimental results show that those two technologies provide enough performance to make Hadoop-based DW feasible. In the future, we plan to expand Hadoop-base DW for advanced analysis, and also integrate interactive analysis and in-stream processing seamlessly for better user experience.

Keywords: Hadoop, Data warehouse, MapReduce, Interactive analysis, In-stream processing

I. 서 론

DW는 전사에 걸쳐 분산 운영되는 데이터를 물리적으로 통합하여 효율적인 관리 및 의사결정을 지원하는 시스템을 말하고 기존 운용 데이터베이스와 비교해 의사결정을 지원할 수 있는 분석정보를 제공한다는 것이 가장 큰 특징이다. DW 시스템의 일반적 구성을 보면 그림 1과 같다[1].

그림 1 에서 보듯이 DW는 다양한 운영계 시스템으로부터 수집한 대규모의 데이터를 사전 정의된 형태로 정제하여 저장소에 보관하고 사용자의 요청에 따라 이를 출력해 주는 역할을 수행하는 전사적 데이터 저장소

논문번호: TR14-023, 논문접수일자:2014.03.19, 논문수정일자:2014.05.07, 논문게재확정일자:2014.05.27
이정룡, 박근태, 장홍성: SK텔레콤

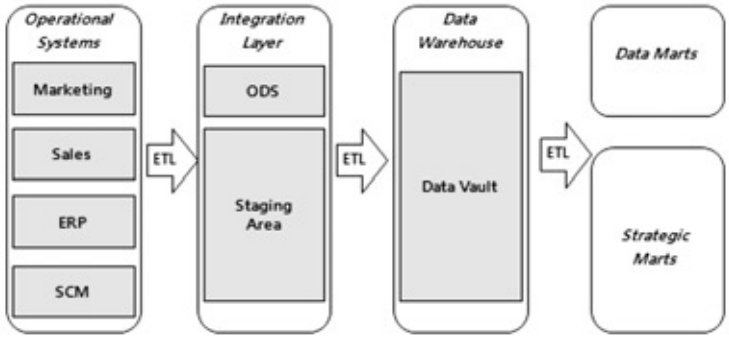


그림 1. DW Overview

표 1. 운영계 DBMS와 분석계 DBMS 비교

특징	운영계 DBMS	분석계 DBMS(DW)
주요 용도	다양한 트랜잭션 처리	데이터 분석
사용자	일반 사용자	관리자, 분석가 등 시스템 관계자
데이터	현재 시점	시계열
작업 단위	트랜잭션	질의
액세스 유형	읽기/쓰기	주로 읽기
액세스 데이터량	수십에서 수백 레코드	수백만 레코드 이상일 경우도 많음
사용자수	수천~수백만명 이상	수십~수백명
성능 척도	Transaction/sec(TPS)	Query/sec(QPS)

이다. 이와 같은 역할을 수행함에 있어 DW 시스템은 역할상 다음과 같이 3개의 Layer로 나눠진다.

① Integration Layer: 다양한 운영계 시스템으로부터 임시 저장소인 Staging Area로 데이터를 수집하고 이를 Operational Data Store(ODS)를 통해 변환, 통합하여 저장하는 역할을 수행한다. 이 같은 작업을 짧은 시간 주기로 수행하기 때문에 ODS로 데이터베이스를 활용하는 경우가 많다. 최근 In-Stream 기술 등의 발전으로 이 주기를 보다 짧게 해서 실시간 처리하려는 시도가 늘고 있다.

② Data Warehouse: 통합된 데이터는 Data Warehouse라고 부르는 또다른 데이터베이스로 이관된다. 이때 데이터들은 Fact나 Dimension과 같은 논리적인 그룹으로 분류되고 구조화 된다. 일반적으로 수십 TB 이상의 큰 데이터를 관리하기 때문에 분산 데이터베이스를 활용한다.

③ Data Mart: Data Warehouse는 대규모 데이터를 운영하므로 Data Warehouse에서 직접 질의나 통계 작업을 수행하는 것은 바람직하지 않다. 이에 Data Warehouse의 데이터를 용도에 맞게 보다 세분화 한 후 Data Mart와 같은 작은 규모의 데이터베이스로 이관해 질의 및 통계 작업을 수행한다.

이처럼 DW는 사용자 질의에 즉시 반응하는 OLTP 기반의 운영계 데이터베이스와는 다른 특징을 요하는데 그 차이점을 비교하면 표 1과 같다.

DW 시스템의 핵심은 읽기 위주의 대규모 데이터에 대해 빠른 질의 응답 성능을 갖는 것인데 이는 Hadoop[2]이 적합한 영역인 Big Data 분석의 특성과 매우 유사하다. DW 분야의 기존 기술들은 대규모 데이터 처리에 한계가 있는데 그 이유는 데이터를 하나 혹은 소수의 서버에서 처리하기 때문에 충분한 성능을 얻기 위해서는 보다 고성능의 CPU, 메모리, 스토리지로 구성되어야 하는 수직적 확장(scale-up) 구조일 수 밖에 없고 이는 고비용일 뿐만 아니라 확장성에 제한이 있기 때문이다. 현재 De-facto 표준으로써 Big Data 분석에 널리 활용되고 있는 Hadoop 기술은 데이터를 수평적 확장이 가능한 다수의 서버에 분산 저장하고 연산을 수행함으로써 기존 DW 시스템 대비 비용 효율적 구조를 갖고 있으며 이 같은 장점을 토대로 Hadoop을 활용해 기존 DW의 일부 또는 전체 기능을 대체하려는 시도가 활발히 진행되고 있다.

1. Integration Layer로서의 Hadoop

Hadoop의 적용으로 데이터 저장 및 ETL(Extract,

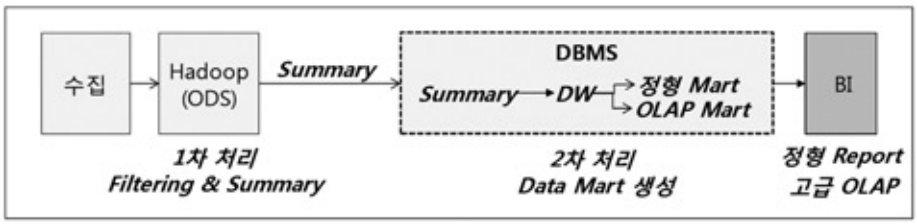


그림 2. Hybrid 방식 DW 구조: Hadoop과 DBMS 결합

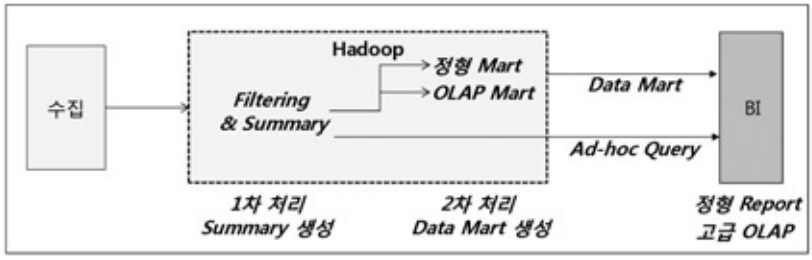


그림 3. Hadoop 기반 DW 구조

Transform, 및 Load), Data Mart 생성, BI tool을 활용한 데이터 분석 작업들 중 대용량의 데이터를 처리해야 하는 Batch 작업인 데이터 저장 및 ETL 작업은 Hadoop에서 수행하고 나머지 Data Mart 생성 및 BI tool 연계 작업만 기존의 DBMS에서 수행하는 것이 가능해졌다. Hadoop을 통하여 충분히 정제된 데이터만 전달되기 때문에 제한된 DBMS 자원으로도 향상된 확장성과 기존 수준의 분석 성능을 사용자에 제공한다(그림 2 참조).

2. Hadoop 기반 DW

데이터에서 새로운 가치를 찾으려는 욕구가 증가함에 따라서 보다 많은 데이터를 BI 기반으로 분석하고자 하는 요구가 나타나고 있는데 이를 충족하기 위해서는 점점 더많은 데이터가 BI Tool과 연계된 DBMS로 전달되어야 한다. 따라서 기존의 낮은 확장성 및 비용 비효율성 문제가 다시 DBMS 단에서 발생하고 이는 Hadoop을 Integration Layer로 활용하더라도 해결할 수 없는 문제이다. 이의 근본적인 해결책으로 데이터의 저장 및 ETL 뿐만 아니라 Data Mart 생성 및 BI tool과의 연계를 통한 분석 과정 전체를 Hadoop 기반으로 수행하려는 연구들이 점차 각광받고 있다[3][4]. 이를 통하면 기존의 정제된 Mart 데이터에 대해서만 BI 분석이 가능했던 구조적 한계를 벗어나서 필요한 경우 raw 데이터 전체에 대해서 고급 분석이 가능한 장점을 얻을 수 있다(그림 3 참조).

Hadoop 기반 DW를 구현하기 위해서는 ① 표준 SQL 기반의 접근 인터페이스를 제공하고 ② 사용자의 요구사항에 따라서 미리 생성하는 비교적 소규모 데이터의 정형 report에 대해서는 기존 DBMS 수준의 빠른 접근 속도를 제공하며 ③ 저장된 대용량 데이터에 대해서 실시간으로 Ad-hoc하게 발생하는 사용자 질의에 대해서도 인터랙티브한 응답을 제공해주는 기능이 필요하다. 하지만 현재의 Hadoop 기술은 위의 요건을 만족시키는데 한계가 있는데 이는 Batch 처리 기반인 Hadoop MapReduce 기술의 다음과 같은 문제점들 때문이다.

3. 디스크 기반 Materialization

MapReduce는 Map과 Reduce 사이 중간 데이터를 디스크에 저장한 후 Reduce 서버들에 분배하여 전달한다. 이를 통하여 Reduce 작업을 수행하는 서버에 장애가 발생하더라도 다른 서버가 해당 역할을 넘겨받아 수행함으로써 전체 작업을 차질없이 완료하는 것이 가능하다. 하지만 디스크 저장시간 소모로 인하여 전체 속도를 저하시키고 더군다나 중간 데이터의 크기가 메모리의 크기를 초과할 경우에는 메모리 크기 단위로 디스크에 저장된 데이터를 하나의 파일로 다시 정렬하여 저장하는 spill 과정에 의해 디스크 I/O가 2배 더 발생한다[5].

4. Map과 Reduce로 단순화된 구조

MapReduce는 사용자 역할을 Map과 Reduce 함

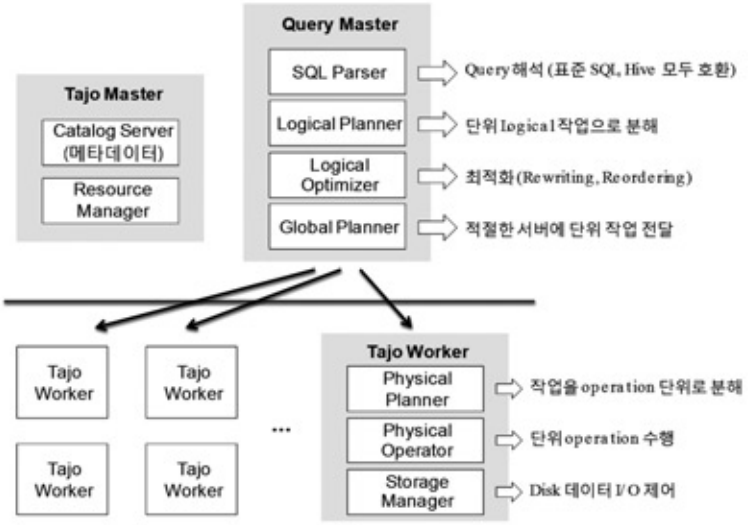


그림 4. Tajo Architecture Overview

수의 구현으로 제한하고 자원관리, 분산처리, 장애 대응 등은 자동 수행하여 높은 사용자 편이성을 제공한다. 하지만 제한된 구현 자유도로 인하여 복잡한 분석 작업은 여러 개의 연속적인 MapReduce로 분해하여 구현해야 하는 단점이 있다. 이러한 이유 때문에 최근의 MapReduce 기반 기계 학습이나 그래프 알고리즘들[6],[7]은 다수의 MapReduce Job을 연속적, 반복적으로 수행함으로써 결과물을 얻을 수 밖에 없다. 따라서, 늘어난 MapReduce 작업의 개수만큼 속도 저하의 원인인 materialization 횟수도 늘어나고 Job들간의 데이터 전달을 위한 HDFS 저장이 추가 발생하여 전체적인 성능은 더욱 저하된다.

5. 높은 초기화 시간

MapReduce는 Job 초기화 단계에 split 수 계산, Task Tracker 상황 점검, 각종 옵션 설정 등을 클러스터를 구성한 전체 서버에 수행한다. 이로 인해 작은 데이터에 대한 계산을 수행하더라도 수십 초의 초기화 시간이 필요하고 이 문제는 클러스터의 규모가 커질수록 더 심해진다.

6. Continuous query

일반적으로 가장 최근에 생성된 데이터가 가장 중요한 경향이 있다. SKT의 경우도 최근 1시간 이내 발생된 데이터 처리에 전체 컴퓨팅 자원의 80% 이상이 활용되고 있고 실시간 분석을 위해 발생 1분 이내 데이터용의 별도 서버 클러스터를 운영하고 있다. MapReduce의 경우 높은 초기화 시간으로 인해 짧은 주기의 지속적인 컴퓨팅에 적합하지 않다. 특히 동일한 데이터에 대한 질의 혹은 동일한 형태의 질의를 반복하는 경우라도 MapReduce 수행에 필요한 모든 초기화 작업을 매번 수행해야 한다. 이 같은 제약사항은 DW의 중요 요소 중 하나인 실시간 ODS를 구현함에 있어 제약이 될 수 있다.

이처럼 현재의 Hadoop MapReduce기술은 높은 응답 지연(high latency)을 보이며 데이터 처리에 대한 즉시성을 보장하기 어려운 한계를 가지고 있다. 이에 대한 해결 방법으로 데이터를 메모리 기반으로 실시간 처리하는 In-stream 기술과 디스크에 저장된 데이터에 대해서도 보다 빠른 분석 및 SQL 기반 분석을 지원하는 Interactive analysis 기술들이 최근에 각광을 받고 있다.

본 논문에서는 위의 요건을 만족하는 Interactive analysis 기술인 Tajo[8]와 오픈소스 기반의 주요 In-stream 기술에 대한 평가 결과를 제시하고자 한다. II장에서는 Tajo의 전반적인 구조 및 SKT 내부 적용을 위해서 기존 대비 개선된 점들을 설명하고 실험 결과를 제시한다. III장에서는 In-Stream 엔진의 구조 및 세부 기술을 살펴본 후 주요 In-Stream 기술들에 대한 기능 및 성능에 대한 실험 결과를 제시한 후 IV장에서 향후 발전 방향을 포함한 결론으로 마무리한다.

II. Interactive Analysis - Tajo

1. Overview

Tajo는 표준 SQL 및 사용자 정의 함수 지원을 통한 기존 DB 시스템과의 호환성을 보장하고 MapReduce

기반 처리의 느린 성능 및 응답속도 문제를 해결하기 위한 유연하고 효율적인 분산처리 엔진과 비용 기반 최적화 엔진 기술을 적용하고 있다(그림 4 참조).

Tajo 클러스터 전체의 자원관리와 테이블 정보관리는 Tajo Master가 담당하고 SQL query 처리는 동적으로 할당되는 Query Master가 담당한다. 각 Query에 대해서 독립적인 Query Master가 존재하기 때문에 Query 처리 간의 간섭을 제거하고 Query Master 역할 서버의 부담도 최소화할 수 있다. Query 처리는 하나의 Query Master와 다수의 Tajo Worker들간의 협업으로 이루어지고 다음과 같은 과정을 따른다[8].

① Parser: 주어진 Query를 해석하여 단위 연산의 트리 형태로 재구성한다.
② Logical Planner: 단위 연산 트리로부터 각 기능 블록의 연관관계를 정의한 Logical node 트리를 도출한다.
③ Logical Optimizer: Join 순서 및 데이터 filtering 적용 순서를 변경하여 최적화된 Logical node 트리를 도출한다.
④ Global Planner: 하나의 Tajo worker 내에서 함께 수행되어야 할 Logical node들을 Grouping한 후 각 worker에 전달하여 수행한다.
⑤ Physical Operator: 각 worker 서버는 할당된 Task를 가장 효율적으로 수행하는 물리적 방법을 선택하여 실행한다.

타조가 기존 대비 우수한 성능을 제공하는 주된 이유는 크게 두가지로 초기 Query의 실행 plan을 결정할 때 Logical Optimizer가 Cost-based join ordering, Projection/Filtering push down 등의 기법을 적용하여 I/O를 최소화하고 실제실행 단계에서는 Progressive optimizer가 생성되는 중간 데이터의 크기에 따라서 Join 순서나 worker에 할당될 task의 개수를 동적으로 최적화하기 때문이다.

SKT에서는 이런 타조를 Hadoop 기반 DW에 적합하도록 OLAP 툴 등 기존의 IT 시스템과의 정합성을 검증하고 대규모 데이터 ETL 작업 속도의 향상 뿐 아니라 ad-hoc query의 빠른 응답을 위한 다양한 기술적인 개선을 진행하였다.

2. 개선된 기능들

2.1. ANSI SQL 지원 향상

DW로서의 기능을 위해서는 Tajo가 SQL 2003 기반의 ANSI-SQL을 완전히 지원하는 것이 필수적이다. 개선을 통하여 함수나 기능면에서 80% 지원 수준까지 도달하였고 '14년 말까지는 90%이상 지원을 목표로 하고 있다(표 2 참조).

또한 상용 OLAP 툴과의 연동을 위한 방법으로 JDBC 드라이버 및 JDBC Proxy 기능을 개발하여 제공한다. JDBC 드라이버는 Tajo 클라이언트로 동작을 하기 때문에 각 Tajo worker에 접속하여 병렬적으로 데이터 I/O를 진행할 수 있다는 장점이 있고 JDBC Proxy의 경우에는 Tajo 클라이언트의 역할을 대행하는 서버를 하나 두어서 사용자가 각 Tajo worker로의 connection을 확보하지 않더라도 Tajo를 활용할 수 있도록 한다. 두가지 모두를 제공함으로써 성능과 네트워크 접근성의 요구사항에 따라서 선택적으로 활용이 가능하도록 하였다.

2.2. ETL 지원 강화

대규모의 데이터에 대한 처리를 효율적으로 하기 위해서 I/O 효율적인 file format들을 추가 지원하고 Batch 처리의 편의를 위한 웹 및 shell 기능이 강화되었다.

2.2.1. 다양한 File Format 지원

소수의 Fact 테이블과 Dimension 테이블들로 구성된 DW의 특성상 row 전체를 읽기 보다는 특정 Dimension과 연계된 column들만을 선택적으로 읽는 것이 성능 향상에 크게 도움이 된다. 이를 지원하기 위해서 Parquet[9], RCFile[10] 등 다양한 Columnar file format들에 대한 지원을 강화하였다.

2.2.2. 웹 및 Shell 기능 강화

Tajo Master의 웹 인터페이스를 통하여 접속할 경우 현재 클러스터의 구성 정보(구성 서버 사양 및 health), 동작 중인 Query 정보(초기 Plan, I/O 크기, 현재 단계별/서버별 진행율), 클러스터 내의 테이블 메타정보 등을 제공한다.

Shell 기반으로도 Query 리스트 확인 및 Kill 작업을 지원하도록 하고 단계별 결과 및 Error 메시지를 표준화하고 주석 및 shell 변수를 지원함으로써 shell 스크립트 기반의 workflow 구성을 보다 쉽게 할 수 있도록 하였다.

2.3. Execution 성능 향상

각 Tajo worker 서버는 할당된 Task에 대해서 시간, 자원 효율성을 극대화하는 방법을 선택하여 실행한다(표 3 참조).

- 데이터의 크기가 메모리 크기보다 작은 경우에는 메모리 기반으로 처리
- 데이터의 양이 많은 경우에는 Hash 기반으로 최대한

표 2. Tajo ANSI SQL 지원

대분류	중분류	소분류	예제	현황
데이터 타입	character string type			○
	numeric type			○
	datetime type			○
select 절	set qualifier		select distinct col1, col2, … from table1	○
	select sublist		select expr1, expr2, … from table2	○
	boolean & numeric expressions			○
table 절	from clause			○
	where clause			○
	group by clause			○
	having clause			○
	order by			○
	Limit			○
from 및 join	relational list join		from a, b, c	○
	qualified join	using condition	tb1 join tb2 using (col1)	'14 예정
		on condition	tb1 join tb2 on tb1.c=tb2.c	○
		(left|right|full) outer join		○
		inner join		○
		natural (left|right|full) outer join		○
		semi join		'14 예정
	table subquery		from (select …) tb1 join (select …) tb2	○
where 절	comparison	〈, 〈 =, 〉, 〉=, =, 〈 〉	col1〈100	○
	predicate			
	between predicate		between 100 and 200	○
	in predicate	value list	in (100, 101, 202)	○
		Subquery	in (select col from table1)	'14 예정
	like predicate		column like 'abc%'	○
	null predicate		col1 is null	○
	qualified comparison predicate		col1〈all (select col2 from table2)	'14 예정
	exists predicate		exists (select … from table1)	'14 예정
	match predicate		col1 (match|unique) (select … from …)	'14 예정
group-by 절	multiple grouping set		group by (a,b,c), (d,e,f)	'14 예정
	Rollup		group by rollup (a,b,c)	'14 예정
	Cube		group by cube (a,b,c)	'14 예정
having 절				○
order-by 절	asc, desc		order by col1 asc, col2 desc	○
	multiple sort keys		order by col1, col2, …, colN	○
DDL 구문	create table			○
	drop table			○
	alter table	rename table		'14 예정
		add column		'14 예정
	function	create function		'14 예정
		drop function		'14 예정

표 3. Tajo Physical Operator 종류

종류	지원하는 Physical operator들
Store	Hash-based column partitioned store, Sort-based column partitioned store
Scan	BST index scan, Partitioned table scan, Sequential scan
Join	Nested loop join, Block nested loop join, In-memory hash join, Sort merge join
Group-by	In-memory hash aggregation, Sort aggregation
Sort	In-memory sort, External sort

분산처리

• Sort 기반으로 결과 데이터의 취합 효율을 극대화

최근에는 CPU에서 Intel SSE[11] 등 데이터의 고속처리를 위한 벡터 기반 연산 instruction을 제공하고 있는데 Java 가상머신은 이런 CPU 특화된 기능을 활용하는데에는 걸림돌이 된다. 따라서 Tajo는 현재는 Java 기반이지만 Physical Operator에서 CPU 고유의 벡터 기반 instruction들을 적극적으로 활용하도록 C++ 기반으로 구현을 진행하고 있다.

SQL 연산의 경우 인자가 가변적인 경우가 많기 때문에(예를 들면 '+' 연산의 경우 integer, big integer, float, double 등의 조합이 가능), 연산의 과정은 ① 인자의 타입을 확인하고 ② 필요한 경우 적절한 형 변환을 선택한 후 ③ 변환을 실행하고 ④ 해당 형에 적합한 CPU instruction 조합을 선택 ⑤ 실행하는 순서로 이루어진다.

테이블 단위로 데이터를 관리하는 SQL의 특성상 동일한 연산을 하나의 데이터값에 대해서만 실행하는 경우보다는 주어진 column 들에 반복적으로 실행하는 경우가 많은데 그런 경우 위의 ①, ②, ④ 과정은 동일한 결과를 얻음에도 불구하고 매번 반복적으로 실행한다. 문제는 ①, ②, ④은 여러 가능한 경우 중 현재 상황과 맞는 경우를 찾는 과정이기 때문에 주로 CPU branch instruction을 수행하게 되고 이는 CPU instruction cache의 효율을 떨어뜨려 전체 수행시간을 증가시킨다.

따라서 ①~⑤까지를 column의 첫 인자에 대해서 수행하였다면 그 이후에 대해서는 ①, ②, ④는 첫번째 인자 계산에서 얻은 값을 바로 활용한다면 전반적인 성능 향상을 가져올 수 있고 이를 위해 Tajo는 수행 과정에 대한 optimization이 가능한 JIT compiler 기술[12]을 활용한다.

2.4. 병목 최소화를 통한 I/O 성능 향상

Hadoop MapReduce와 Tajo 모두 어플리케이션 수준에서 분산 병렬처리를 수행하는 Framework이기 때문에 처리하는 Query의 수가 증가하면 OS 수준에서 관리하는 디스크에서는 여러 Query 처리에 필요한 I/O들에 의해 병목 현상이 발생한다. 특히나 서버(locality-aware)나 랙(Rack-aware)은 구분하지만 하나의 서버 내에 다수의 디스크에 대해서는 구별하여 작업을 시작하지는 않기 때문에 단순 메모리 크기나 CPU core 수에 따라서 작업을 분배한 경우 다수의 작업이 하나의 디스크로 몰리는 현상이 더욱 빈번해진다.

따라서 우선은 Tajo에 disk-awareness를 도입하여 하나의 Tajo worker에 할당된 작업들은 가급적이면 서로 다른 disk에서 진행하도록 하였고 하나의 disk 상의 작업에 대해서도 Storage Manager를 통하여 여러 Query가 해당 디스크에 순차적으로 접근하도록 함으로써 병목에 의한 비효율을 최소화한다(그림 5 참조).

- 디스크에 대한 I/O 요청은 Storage Manager의 Scheduler를 무조건 거쳐 각 디스크별 queue를 통하여 순차적으로 실행된다.
- 하나의 I/O 요청은 bulk 단위로 처리하여 디스크의 I/O 효율을 극대화 한다.
- 단위 I/O 크기에는 제한(현재 최대 8MB)을 두어 request queue에서 HOL(Head of line) blocking 처럼 하나의 요청에 의해 다른 요청의 수행시간이 지연되는 문제를 방지한다.

3. 성능 평가

3.1. 테스트 환경

SQL-on-Hadoop 중 가장 널리 쓰이는 Hive (version 0.12)[3] 및 최근 들어서 in-memory 처리를 통하여 Hive를 개선한 기술인 Impala(version 1.3.0)[13]와 비교하여 데이터베이스 관련 벤치마크 중 하나인 TPC-H(Revision 2.16.0)[14]를 활용하여 성능을 측정하였다(표 4 참조).

- ANSI SQL 호환이 완벽하지 않은 Hive의 경우 모든 Query의 형태를 HiveQL로 변환하였고 Impala의 경우에도 해당 기술에서 제공하는 수정된 SQL을 활용하였다. Tajo의 경우에는 ANSI SQL을 지원하기 때문에 Query를 변형없이 수행하였다.
- TPC-H는 100GB의 데이터를 기준으로 하였고 caching 효과를 제거하기 위해서 매 테스트는 cache clear 후 수행하였다.
- 각 테스트는 Hive, Tajo, Impala를 번갈아 가며 수

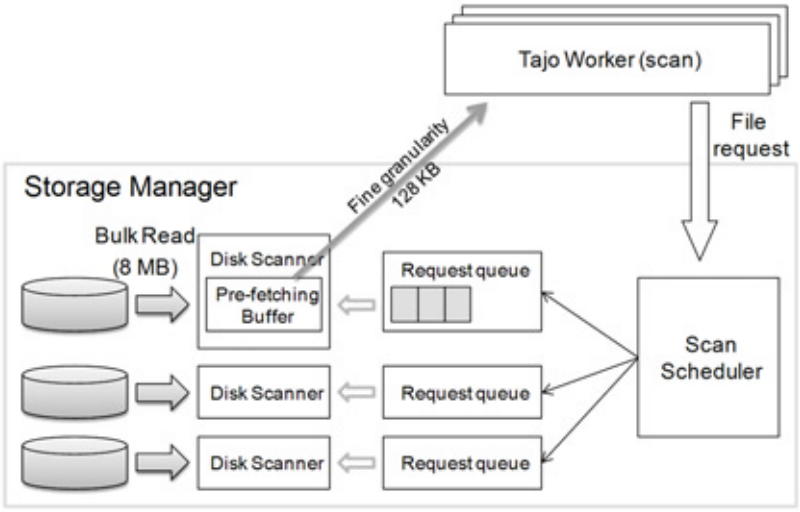


그림 5. Tajo Storage Manager

표 4.

서버 구성	
Hadoop 구성	Namenode x 1, Datanode x 6
Tajo 구성	Master x 1, Worker x 6
서버 Spec.	
CPU	Xeon E5-2640 (2.50GHz, 6cores, HT) x 2ea
Memory	64GB
Disk	3TB(7.2k RPM) x 6ea
Network	동일 rack 내에서 1G Ethernet 스위치를 통하여 상호 연결

행하였고 각각을 10번 측정하여 그 평균을 구하였다.

3.2. 결과

3.2.1. 전반적 성능 비교

TPC-H 실행 결과 Tajo가 Hive 대비 약 5배 성능 향상을 보이고 Impala에 비해서는 90%의 성능을 보인다. Hive 대비 성능 증가는 최적화된 Query 실행 Plan 및 병목을 최소화 한 효율적인 디스크 I/O부터 오고 이를 통하여 In-memory 기반으로 동작하는 Impala에도 비견할 만한 속도를 보인다. Tajo가 서버 장애시 Query 처리 안정성 및 처리 가능한 데이터 용량의 대폭적인 확대를 위해 Disk 기반 materialization을 활용하고 있음을 고려하면 Tajo의 성능 향상 효과가 Memory와 Disk 간의 성능 간극을 메워줄 수 있을 정도로 큼을 알 수 있다(그림 6 참조).

분석에 활용되는 field가 비교적 적어 계산이 상대적으로 단순한 Q3, Q4, Q6, Q14, Q16, Q19 보다는 다차원 분석을 모델링한 Q2, Q10, Q15, Q18가 더 좋은 성능 향상을 보여준다. 이는 단순 계산 Query는 Disk I/O가 주가 되는 연산(Disk-intensive)을 하는 반면에 다차원 분석은 계산이 주가 되는 연산(CPU-intensive)을 하기 때문으로 CPU-intensive의 경우 디스크 효율 향상 뿐 아니라 실행 Plan 효율 개선, Physical Operator 개선의 효과 모두를 복합적으로 볼 수 있기 때문이다. 따라서 Disk 연산이 주가 되는 Q1, Q8, Q12, Q14, Q17, Q19의 경우에도 상대적으로 낮은 성능 향상을 보임을 볼 수 있다.

3.2.2. Disk-intensive

디스크 I/O가 주가 되는 작업의 경우 디스크의 물리적 속도가 전체 성능의 한계가 되기 때문에 엔진의 성능 향상 효과가 반감된다. 하지만 이러한 경우에도 Tajo는 Hive 대비 3배 이상의 성능을 보이는데 이는 Tajo의 Storage Manager가 서버 디스크별 I/O 작업을 최대한 병목없이 처리하기 때문이다.

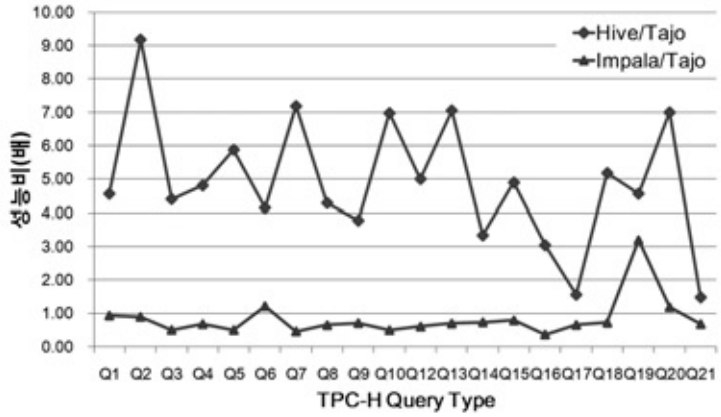


그림 6. Tajo와 Hive의 TPC-H 성능 비교

표 5. Storage Manager에 의한 성능 향상

# of concurrent workers	Disk throughput (MB/s)	
	No Storage Manager	Storage Manager
4	72.4	72.8
10	27	69.6

표 5는 Storage Manager 적용 전후의 디스크 Throughput의 변화를 측정해 본 것으로 디스크에 동시 접근하는 worker thread의 개수가 10개가 되면 2.5배 이상의 성능 차이가 남을 볼 수 있다. 하나의 클러스터가 다수의 사용자 혹은 Job에 의해서 공유되는 Hadoop 환경 하에서는 각 디스크로 동시 접근하는 worker thread의 수가 더욱 늘어나기 때문에 storage manager에 의한 더 큰 성능 향상 효과가 기대된다.

Ⅲ. In-Stream data Processing - 주요 In-Stream 엔진 평가

1. Overview

최근의 ODS 및 Data Composite 소프트웨어들은 실시간 ETL을 지원할 만큼 데이터에 대한 실시간 처리는 매우 중요해지고 있다. Hadoop MapReduce 나 Tajo와 같은 IA 기술을 DW 분야에 활용함에 있어 갖는 문제점 중에 하나는 실시간으로 빠르게 수집되는 데이터 또는 이벤트에 대해 즉각적인 처리가 어렵다는 점이다. In-stream data processing 은 실시간으로 유입되는 데이터에 대해 즉시 처리하는 기술로 MapReduce 나 IA와 같은 배치 기반의 시스템의 단점을 보완할 수 있다. In-Stream 기술은 Spam detection, 공격 탐지 등과 같은 보안 분야, 시스템 모니터링 분야, 실시간 인기 검색어 같은 분야에서 활발하게 사용되고 있고 최근에는 fraud detection 이나 sketch 알고리즘을 활용한 패턴 예측과 같은 복잡한 분야에서도 활용되고 있다.

In-Stream 엔진은 Stream 데이터 처리 방식에 따라 한번에 하나의 이벤트를 처리하는 Record at a time 방식과 일정 시간 간격으로 일정간격으로 다수의 Event를 모아 한번에 처리하는 Discretized Stream 방식[15]으로 분류할 수 있다. 이들 두가지 방식은 동작 방식에 차이는 있으나 In-Stream 엔진으로서 갖춰야 할 구성요소는 유사하며 그림 7과 같이 In-Stream 엔진의 일반적 구성요소를 정리할 수 있다[16].

① Broker: 데이터를 수집하고 일시적으로 보관하는 곳으로 In-Stream 엔진의 진입점이 된다. 대량의 데이터를 처리 가능하고 Fault-tolerant하며 필요시 유입된 In-Stream 데이터를 처리엔진에 다시 전달하는 Stream replay 기능을 지원한다. 과거 Broker의 경우 Queue 저장소로 단일 SAN이나 NFS v4 등을 많이 사용해 왔으나 현재는 데이터량 증가에 대한 대응과 비용 절감을 위해 분산 데이터베이스에서 주로 활용되는 Partitioning, Replication 등의 기술을 추가로 활용하고 있다. 이로 인해 분산처리 엔진의 일부 기능을 Broker가 포함하는 경우가 많은데 In-Stream 처리에서 많이 활용되는 오픈소스 Broker 중 하나인 Kafka의 경우 MapReduce Framework과 유사한 Hash 기반

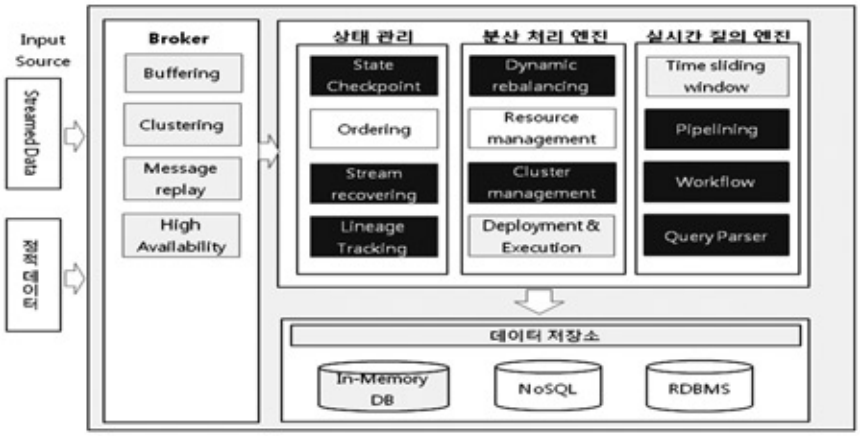


그림 7. In-Stream 엔진 구성

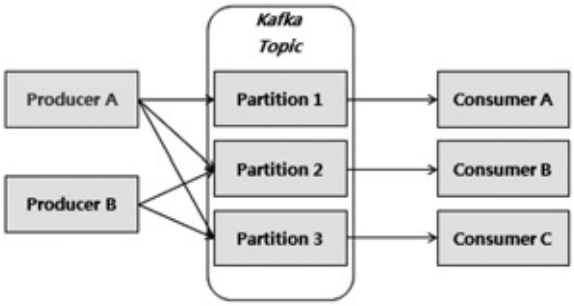


그림 8. Kafka Partition

Partitioning을 제공함으로써 In-Stream 엔진의 부하를 줄일 수 있다[17](그림 8 참조).

② 상태관리: 상태관리는 입력된 데이터가 잘 처리 됐는지를 보장하기 위한 기능이다. 유입된 데이터를 유입 순서대로 처리되는 것을 보장하고(Ordering), 복잡한 플로우 모두를 성공적으로 처리했는지 확인한다(Lineage Tracking). 또 오류가 발생했을 때 다시 시도할 수 있어야 하며(Stream recovering) 무한히 유입되는 데이터에 대해 in-Stream의 상태를 외부 저장 매체에 보존해 줘야 한다(State checkpoint). 이와 같은 기능은 Transactional 시스템에서 매우 기본적인 기능이나 분산된 환경에서 거대한 데이터를 다룰 때는 결코 사소하지 않다. 특히 Lineage Tracking, State checkpoint의 경우 In-Stream 엔진에서 각 Event 단위로 상태를 관리해야 하므로 구현시 성능에 문제가 발생할 수 있다. Storm[18] 의 경우에는 다수의 Event를 하나의 그룹으로 묶은 Transaction id 단위로 상태를 관리하며 처리 실패 시도 해당 단위로 재시도를 수행함으로써 상태관리를 위한 부하를 줄인다[19].

③ 분산처리 엔진: 분산처리 엔진은 Hadoop의 MapReduce와 크게 다르지 않다. 작업을 배포하고 실행하며(Deploy & Execution) 리소스 상황에 맞게 작업에 컴퓨팅 자원을 할당 및 관리한다(Resource Management). 분산처리 중 일정 부분이 실패하면 다른 노드에 작업을 할당하고(Dynamic Rebalancing) Master와 Worker등 전체 분산 컴퓨팅 자원을 관리한다(Cluster Management). 이와 관련하여 최근에는 범용적 클러스터 자원관리 시스템인 YARN, Mesos 등과 연계하려는 노력이 진행되고 있다. 하지만 이들은 아직은 MapReduce 등 배치 처리용 엔진에 최적화 되어 있어 In-Stream 엔진에서 최적의 성능을 발휘하지 못한다[20]. SKT에서는 YARN과 In-Stream 엔진의 통합을 위해 In-Stream 엔진에 맞는 Application Master 의 개발을 진행하고 있다.

④ 실시간 질의 엔진: 실시간 질의 엔진은 SQL과 같은 DSL(Domain Specific Language)을 분산 처리 엔

표 6. In-Stream 엔진 기능 평가

	Storm	Spark Streaming	Samza	Esper
Data 처리 방식	Record at a time	Batch	Record at a time	Record at a time
Query Engine (pipelining)	API 방식	API 방식	API 방식	EPL
Ordering & Guarantee	부분 지원	지원	부분 지원	부분 지원
State Management	사용자 정의	Built-in	Built-in	Built-in
Dynamic rebalancing	지원	지원	미지원	미지원
Deployment & Execution (Resource management)	자체 엔진 YARN	자체엔진 YARN	YARN	자체(상용만 지원)
Language support	Java외 다수	Java, Scala, Python	Java	Java, .NET
Workflow	지원	지원	미지원	미지원
Maturity	높음	낮음	낮음	매우 높음
Buffering	ZeroMQ	Memory+Disk	Disk	자체 Queue
Data Model	Tuple	RDD	Java serialization	Java serialization
Development Support (로컬 테스트 환경)	지원	지원	미지원	지원
Performance (Messages/node/sec)	115k+	600k+	80k+	20k+

진이 처리할 수 있는 작업 형태로 전환해 주는 역할을 수행한다. 앞서 설명한 Interactive Analysis 기술인 Tajo의 Parser와 같은 역할을 수행한다. 차이점은 실시간 질의 엔진은 Window 기능을 지원한다는 점으로 영구히 유입되는 데이터를 모두 처리할 수 없으므로 특정한 시간 범위 안에서만(예: 최근 30분) 질의를 수행한다.

이와 같은 기능 요소를 토대로 현재 공개된 대표적인 오픈소스 In-Stream 엔진들에 대한 기능 및 성능 평가를 수행하였다. 우선 각 엔진의 기능성을 평가하여 적합한 기능을 보유한 엔진들을 선정한 후 이에 대한 성능 평가를 수행하는 방식으로 진행하였다.

2. 기능 평가

앞서 언급한 In-Stream 엔진의 기본 요소 기능에 사용성, 확장성을 추가 검토 정리했다(표 6 참조).

각각은 모두 전체적으로 기본 기능들을 만족한다. 각 엔진에 대한 장단점을 간략히 살펴보면 다음과 같다.

① Storm[18]: 오픈소스 중에는 가장 성숙된 엔진이며 수많은 커뮤니티로부터 지원을 받고 있다. 특히 트랜잭션 처리를 위한 별도의 프레임워크(Trident)를 제공하며 초당 10만 이상의 Event를 처리한다. 하지만 상태관리를 사용자가 직접 구현해줘야 하는 불편이 존재하며 Tuple 단위 통신으로 인해 네트워크 부하가 높은 편이다. 또한 Cluster Controller인 Nimbus가 Single Point of Failure(SPOF)로 장애에 취약하다.

② Spark[21]: 한번에 하나의 Event를 처리하는 Record at a time 방식의 In-Stream 엔진들과는 달리 다수의 Event를 모아 한번에 처리하는 연속적인 짧은 Batch 방식을 이용한다. In-memory 컴퓨팅 엔진인 Spark을 In-Stream에도 활용한 형태로 Batch 방식을 사용함으로써 Fault-tolerance, Stream Replay가 우수하며 Event 처리 Throughput이 뛰어나다. 하지만 아직 기술적 성숙도가 떨어지며 상태저장을 위한 인터페이스가 부족하며 Record at a time 방식에 비해 Latency가 높다.

③ Samza[22]: 전체적인 기능에 있어 Storm과 매우 유사하다. 차이점은 Job당 하나의 CPU Core만을 사용함으로써 자원관리에 용이하고 상태관리를 제공하며 경량의 RPC 프로토콜을 지원함으로써 네트워크 부하를 줄일 수 있다. 하지만 Spark과 마찬가지로 아직 성숙도가 낮고 로컬테스트를 위한 환경을 아직 지원하지 못하며 현재 개발된 ApplicationMaster의 경우 이중화를 지원하지 못해 장애에 취약하다.

④ Esper[23]: Esper는 비교 대상 엔진 중 가장 성숙된 In-Stream 엔진으로 EsperTech 사에 의해 지원되며 다양한 레퍼런스를 가지고 있다. Esper는 비교 대상 엔진 중 유일하게 질의를 할 수 있는 EPL을 제공하며 매우 복잡한 질의도 처리할 수 있다. 다만 다른 엔진에 비해 처리 성능이 낮고 이중화, 분산처리와 같은 확장을 위한 기능은 상용 버전에서만 지원된다. 오픈소스에서 확장성을 해결하기 위해 Storm내에 Esper를 활용하는 방안도 제시되고 있다.

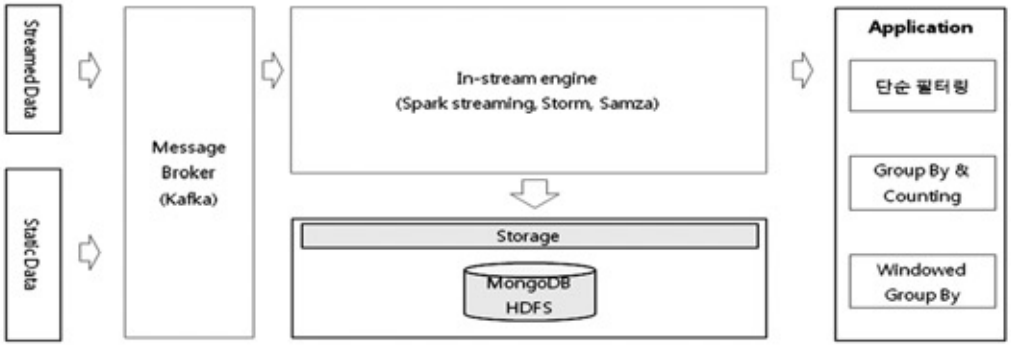


그림 9. 성능 평가 Architecture

표 7.

서버 구성 (총 20 노드)	
CPU	Xeon E5-2640 (2.50GHz, 6cores, HT) x 2ea
Memory	64GB
Disk	4TB(7.2k RPM) x 4ea
Network	1Gigabit dual bonding channel 20 node

표 8. Grep 결과

	200 Bytes	600 Bytes
Spark Streaming(Batch Interval 5s)	37MB/s/node	38MB/s/node
Spark Streaming(Batch Interval 2s)	32MB/s/node	33.5MB/s/node
Storm	13MB/s/node	19MB/s/node

표 9. Word Count 결과

	200 Bytes	600 Bytes	600 Bytes
Spark Streaming (Batch Interval 5s)	27MB/s/node	27.1MB/s/node	27.1MB/s/node
Spark Streaming (Batch Interval 2s)	25MB/s/node	25MB/s/node	25MB/s/node
Storm	9MB/s/node	11.2MB/s/node	동작실패

이상의 4가지 오픈소스 In-Stream 엔진에 대해 기능, 사용성, 확장성을 평가한 결과, 기능성 및 성숙도가 부족한 Samza와 확장성 및 성능이 부족한 Esper는 대상에서 제외하고 Storm, Spark 두가지 엔진에 대해 성능 평가를 수행하였다.

3. 성능 평가

성능 평가를 위한 서버 사양은 아래와 같고 In-Stream 엔진에는 16GB의 메모리를 할당하였다. 이와 같은 서버 환경에서 레코드당 200, 600 바이트인 데이터를 이용해 성능 테스트를 진행했다(표 7 참조).

기본 데이터 구성 아키텍처는 다음과 같다(그림 9 참조).

- 200, 600 바이트 레코드에 대해 각각 특정 단어에 대한 Grep, 1분 Windowed time안에서 Word Count를 수행하였다(표 8 참조).
- Spark Streaming의 경우 Batch Operation을 활용하는 관계로 Batch Interval을 설정해야 하는데 여기서는 Batch Interval을 2초, 5초로 설정해서 테스트를 수행했으며 Partition의 수는 128로 고정시켰다.
- Word Count 테스트의 경우 Spark Streaming에서만 Combiner 클래스를 통한 네트워크 사용량을 줄

이는 방법을 선택했다(표 9 참조).
- 각기 10분간 10회에 걸쳐 테스트를 수행했으며 각 결과에 대해 평균값을 냈다.

전체적으로 Spark Streaming의 성능이 Storm 보다 앞서는 것을 확인할 수 있다. Record at a time 방식에 비해 Buffering을 통해 한번에 처리하는 Batch 방식이 성능적 장점을 가지고 있기 때문이다. Storm의 경우에도 record의 길이 증가시 한번에 처리하는 데이터량이 증가하기 때문에 성능이 점차 증가하는 것을 볼 수 있지만 네트워크 전송량도 함께 증가하므로 record의 길이 증가에 따른 효과는 제한적일 수 있다. 실험 결과에서도 1000 Byte 크기의 record를 사용하는 경우 네트워크 I/O의 병목이 발생함으로써 테스트를 수행할 수 없었다. 반면에 Spark Streaming의 경우에는 Transformation 단계에서 Combiner 를 활용함으로써 노드간 네트워크 트래픽이 감소하여 이러한 문제를 회피할 수 있었다.

Spark Streaming의 경우 Batch Interval의 값이 작아질수록 처리량이 감소하는 것을 확인할 수 있다. 이는 Batch Interval이 작아질수록 Batch 처리를 준비하는 오버헤드를 무시할 수 없기 때문으로 Batch Interval이 작아질수록 분석의 Latency는 작아지지만 Throughput 역시도 작아진다. 따라서 Latency와 Throughput 요구 수준에 맞는 적합한 Batch Interval을 설정하는 것이 중요하다.

IV. 결 론

Big Data 기술은 기존 DW의 한계인 시스템 확장성이 제한되고 분석 대상이 요약된 데이터로 한정되는 문제를 해결하기 위한 핵심 기술로서 자리 매김하고 있다. 본 논문은 여기에서 한발 더나아가 이러한 Big Data 기술이 DW의 전체 기능을 대체 가능하도록 하는 Interactive Analysis 기술과 In-stream processing 기술을 설명하였다. SKT에서는 위의 기술을 활용하여 수백 TB 규모 Hadoop 기반 DW의 구축을 진행 중이고 본 논문에서 살펴본 것처럼 성능, 확장성, 비용 효율성 측면에서 기존 DW를 크게 개선하리라 예측된다.

향후에는 Hadoop 기반으로 완벽하게 대체하지 못하고 있는 고급 분석을 Big Data 처리에 포함하려 하고 이를 위해 Tajo 내부의 Analytic function 확충 및 Tajo의 병렬처리 성능을 활용한 R, python, OLAP connector 개발을 진행 중이다. 또한 In-Stream 엔진을 활용하기 위해서는 해당 엔진이 제공하는 API를 활용해 사용자가 직접 개발을 해야 하므로 일반 사용자의 접근성이 떨어지는 문제가 존재하는데 이의 해결을 위해 Tajo의 SQL 처리 기술과 In-Stream의 접목을 추진할 예정이다.

〔참고문헌〕

[1] Wikipedia, "Data warehouse," [Online]. Available: http://en.wikipedia.org/wiki/Data_warehouse.

[2] [Online]. Available: http://hadoop.apache.org.

[3] Thusoo and Ashish, "Hive-a petabyte scale data warehouse using hadoop," in *26th International Conference on Data Engineering (ICDE)*, 2010.

[4] S. Chen, "Cheetah: a high performance, custom data warehouse on top of MapReduce," in *Proc. VLDB Endow. 3*, Sep. 2010.

[5] T. White, Hadoop: The Definitive Guide, O'Reilly Media, 2009.

[6] [Online]. Available: https://mahout.apache.org.

[7] U. Kang, C. E. Tsourakakis and C. Faloutsos, "Pegasus: A peta-scale graph mining system implementation and observations," in *Ninth IEEE International Conference on Data Mining (ICDM)*, 2009.

[8] H. Choi, J. Son, H. Yang, H. Ryu, B. Lim, S. Kim and Y. D. Chung, "Tajo: A distributed data warehouse system on large clusters," in *29th IEEE International Conference on Data Engineering (ICDE)*, 2013.

[9] "Parquet," [Online]. Available: http://parquet.io.

[10] Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang 그리고 Z. Xu, "RCFile: a fast and space-efficient data placement structure in MapReduce-based warehouse systems," %1 *27th IEEE International Conference on Data Engineering (ICDE)*, 2011.

[11] C. Lomon, "Introduction to Intel advanced vector extensions," Intel White Paper, 2011.

[12] M. Zukowski and J. Sompolski, "Just-in-time Compilation in Vectorized Query Execution," 2011.

[13] M. Kornacker and J. Erickson, "Cloudera Impala: real-time queries in Apache Hadoop, for real," 10 2010. [Online]. Available: http://blog. cloudera. com/blog/2012/10/cloudera-impalareal-time-queries-in-apache-hadoop-for-real.

[14] TPC, "TPC-H benchmark," [Online]. Available: http://www.tpc.org/tpch/.

[15] Z. Matei, D. Tathagata, L. Haoyuan, H. Timothy, S. Scott and S. Ion, "Discretized streams: A fault-tolerant model for scalable stream processing," in *UC Berkeley Technical Report UCB/EECS-2012-259*, 2012.

[16] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. C. etintemel, Y. Xing and S. Zdonik, "Scalable Distributed Stream Processing," in *CIDR Conference*, Asilomar, CA, 2003.

[17] "Kafka 0.8.1 Documentation," [Online]. Available: http://kafka.apache.org/documentation.html.

[18] [Online]. Available: http://storm.incubator.apache.org/.

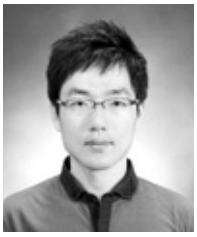
[19] "In-Stream Big Data Processing," 20 8 2013. [Online]. Available: http://highlyscalable.wordpress.com/2013/08/20/in-stream-big-data-processing/.
[20] C. Byung-Gon, C. Tyson, C. Carlo, D. Chris, M. Sergiy, M. Brandon, N. Shravan, R. Raghu, R. Sriram, R. Josh, S. Russell and W. Markus, "REEF: Retainable Evaluator Execution Framework," in *VLDB*, 2013.
[21] [Online]. Available: http://spark.apache.org.
[22] [Online]. Available: http://samza.incubator.apache.org/.
[23] [Online]. Available: http://esper.codehaus.org/.

장 홍 성
(Hongsung Chang)

1992: 서울대 컴퓨터공학과 학사
1994: 서울대 컴퓨터공학과 석사
1999: 서울대 컴퓨터공학과박사
1999.3~2005.7: 삼성전자 정보통신총괄 책임연구원
2005.8~현재: SK텔레콤 N/W기술원, IT기술원, 성장기술원 랩장
관심분야: Big Data, Hadoop, Cloud computing, ICT Convergence
E-mail: hschang7@sk.com
Tel: +82-2-6100-5321

이 정 룡
(Jungryong Lee)

2001: 연세대 화학공학과 학사
2001. 3~2013. 9: 삼성 SDS
2013. 9~현재: SK텔레콤 성장기술원
관심분야: Big Data, Hadoop, Distributed Computing System
E-mail: i1befree@sk.com



박 근 태
(Keuntae Park)

1999: KAIST 전기 및 전자공학 학사
2001: KAIST 전기 및 전자공학 석사
2007: KAIST 전기 및 전자공학 박사
2007. 3~2008. 12: 한국전자통신연구원
2008. 12~2010. 3: TmaxCore
2010. 5~현재: SK 텔레콤 기반기술연구원, IT 기술원, 성장기술원
관심분야: Big Data, Tajo, O/S, Network, Distributed Computing System
E-mail: keuntae.park@sk.com