ISBN XXX-XX-XXX-XXXX-X-XXXXX

보안이벤트 수집 성능 향상을 위한 NoSQL의 빅데이터 처리 성능 비교

2016. 11

KiSTi www.kisti.re.kr
한국과학기술정보연구원
Korea Institute of Science and Technology Information

보안이벤트 수집 성능 향상을 위한 NoSQL의 빅데이터 처리 성능 비교

2016. 11

부　서 : 첨단연구망센터 첨단연구망정보보호실
제출자 : 정용환 (paul7931@kisti.re.kr)
최장원 (jwchoi@kisti.re.kr)
이행곤 (hglee@kisti.re.kr)

KiSTi www.kisti.re.kr 한국과학기술정보연구원 Korea Institute of Science and Technology Information

[목 차]

제1장 서론

최근 스마트폰 및 타블렛 PC 등 모바일 디바이스의 보급과 함께 국내 인터넷 이용자 수가 4,194만 명에 이르렀다. 인터넷 이용자 수의 증가에 따라 인터넷 사용량도 폭발적으로 증가하였고, 악의적인 침해위협 시도 또한 점진적으로 증가하여 많은 양의 보안이벤트가 발생되고 있는 추세이다.

기존의 침해위협 탐지시스템은 RDB 기반으로 구성·운영되고 있기 때문에 빅데이터 환경에서 RDB 기반의 보안이벤트 처리는 전체 시스템의 잠재적인 병목 구간으로 판단되며, 실시간 보안관제의 관점에서는 치명적인 요소로 작용할 수 있기 때문에 보다 신속한 데이터 처리가 가능한 대안이 필요한 실정이다.

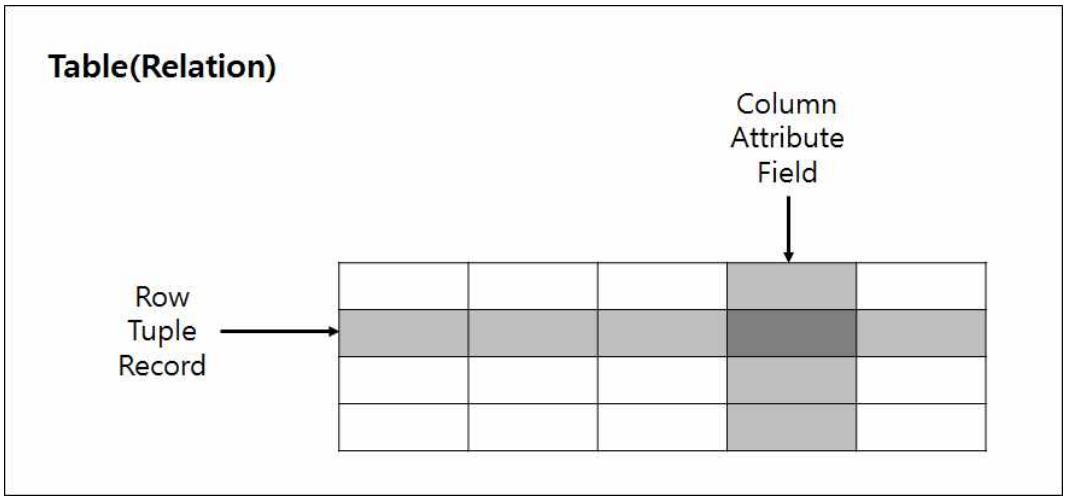
최근 Cassandra, HBase, MongoDB 등 비관계형 데이터베이스(이하 NoSQL)와 Hadoop, 아파치 Storm 등 분산 플랫폼 등이 빅데이터 처리 기술로 많이 활용되고 있는 추세로, 본 문서에서는 빅데이터화 되고 있는 보안이벤트 처리 관점에서의 이러한 기술들의 적용 가능성을 알아보고자 한다.

제2장 관련 연구

제1절 관계형 데이터베이스(RDB)

관계형 데이터 모델은 1970년대에 Codd, Edgar. F 박사에 의하여 발표된 데이터 모델의 하나로 수학의 집합 개념을 기초로 한 데이터베이스 모델이다. RDB는 행(Row)과 열(Column)로 이루어진 관계(Relation)와 수학적으로 정의된 연산들로 구성된다. RDB는 행을 레코드(Record)란 용어로, 열을 필드(Field)란 용어로 그리고 Relation을 테이블(Table)이란 용어로 사용한다.

RDB는 테이블 형태로 개체(Entity)의 모든 데이터와 데이터 사이의 관계를 기술한다. 테이블은 행과 열로 구성되어 있으며 테이블의 각 행은 Entity의 인스턴스를 가리키는 레코드 또는 튜플(Tuple)들로 구성된다. 열은 한 개 이상의 필드로 구성되며 각 열의 필드들을 Attribute(Attribute) 또는 속성이라고 한다. <그림 2-1>은 RDB의 테이블 구성을 나타낸다.



<그림 2-1> Table Configuration of RDB

테이블의 각 행은 관련된 데이터의 집합을 나타내며, 한 테이블 내에서 모든 행은 동일한 구조를 가지게 된다. 열은 행에서 각각의 필드들이 어떠한 속성을 가져야 하는지를 결정한다. 또한, 서로 다른 테이블 간의 관계를 정의 할 수 있으

며 대부분의 RDB는 데이터베이스를 관리하기 위해서 SQL을 사용한다.

데이터베이스에서 각 Entity의 관계에서 다른 Entity에서 그 Attribute의 속성값과 같은 값을 갖지 못할 때 Attribute의 값이 유일한 값을 가질 경우 이 Attribute를 Entity 식별자라고 하며, Attribute에 대응하는 데이터 항목을 기본 키(Primary Key)라고 한다.

파일 관리에서 이 파일을 검색할 때 검색의 수단으로 하는 데이터 항목을 키 항목이라 하고, 키 항목에 의하여 Entity의 식별이 가능할 때, 즉 레코드 식별이 가능할 때 이 키를 기본 키라고 한다. 모든 관계가 단일 속성의 기본키를 갖는다고는 볼 수 없지만, 모든 관계는 몇 개의 속성을 연결시켜서 유일성을 가질 수 있는 속성의 조합이 가능하다. 단일 속성으로 구성되는 조합은 단지 특수한 경우에 지나지 않는다.

< 키의 종류 >

■ **후보 키 (Candidate Key)**

어떤 관계 R의 모든 튜플에 대한 최소성과 유일성을 만족하는 키를 후보 키라고 한다. 유일성은 어떤 관계의 각 튜플을 유일하게 구분 지을 수 있는 성질을 말하며, 최소성은 유일성을 파괴하지 않고 기본키에 속한 속성의 일부를 제거할 수 없는 성질을 말한다. 모든 릴레이션은 적어도 하나의 후보키를 반드시 가지고 있다.

■ **기본 키(Primary Key)**

기본 키는 주어진 어떤 관계에서 그 튜플 내의 유일한 값을 가진 튜플을 유일하게 식별하는데 사용되는 키를 말한다. 기본 키는 후보 키 중에서 주로 사용 되는 키 Attribute를 선정하여 기본 키로 지정한다. 즉, 기본 키는 후보 키 중에서 주로 튜플을 유일하게 식별할 수 있는 Attribute의 집합을 기본 키로 지정할 수 있다. 기본 키는 유일무이한 존재이고 최소성을 가지고 있어야 한다. 이러한 이유에서 기본 키의 값은 Null 값을 가질 수 없다. Null 값은 아직 알려지지 않은 값이나 해당 없음 등의 이유로 정보 부재를 나타내는 특수한 데이터 값으로 공백과는 다른 특수한 값이다.

■ 대체 키(Alternate Key)

대체 키는 역시 최소성과 유일성을 만족하면서 기본 키의 역할을 대신할 수 있는 키를 말한다. 대체 키는 후보 키 중에서 기본 키로 선정되지 않은 키를 대체 키로 사용한다.

■ 대체 키(Alternate Key)

대체 키는 역시 최소성과 유일성을 만족하면서 기본 키의 역할을 대신할 수 있는 키를 말한다. 대체 키는 후보 키 중에서 기본 키로 선정되지 않은 키를 대체 키로 사용한다.

■ 슈퍼 키 (Super Key)

슈퍼 키는 유일성만 갖고 최소성이 없는 키를 말한다.

■ 외래 키 (Foreign Key)

어떤 관계 R1에 정의되어진 Attribute 집합을 FK라고 할 때, 임의로 주어진 시점에서 R1의 FK 값이 어떤 관계 R2에 정의되어진 기본 키가 된다라고 할 때 이 FK는 관계 R1의 외래 키라고 한다. 이 경우에 Attribute 집합 FK는 관계 R2를 참조한다라고 하며, 이 때 관계 R2는 참조 관계라고 한다. R1의 FK와 R2의 기본 키에 속한 Attribute 이름은 반드시 같을 필요는 없다. R1의 FK가 R1의 기본 키이거나 기본 키에 포함될 필요는 없다.

1. 관계형 데이터베이스 제약

RDB에서 관계 데이터 제약은 데이터의 무결성을 지키기 위하여 제약조건을 사용한다. 즉, RDB에서 관계 데이터 제약은 데이터베이스에 저장되는 데이터에 대한 규칙을 의미한다. 이는 데이터베이스가 기본 구조를 유지하기 위해 데이터를 정확하게 표현하고 오류를 방지하기 위해 필요하다. 본질적 제약은 데이터 모델의 구조적인 특성 때문에 생기는 제약, 즉 데이터베이스의 기본 구조를 유지하기 위해 필요하다. 그리고 내재적 제약은 데이터의 의미를 정확히 표현하고 오류를 방지하기 위하여 데이터베이스 스키마에 구체적으로 내재시키는 제약을 말한다. 그러나 모든 제약을 데이터베이스로 표현한다는 것을 불가하기 때문에 데이터베이스 스키마에 표현할 수 없는 제약은 프로그램에 명시하거나 사용자의 수작업에 의존하게 된다. 이를 명시적 제약이라고 한다.

기본적인 관계 데이터 제약은, 영역 무결성 제약, 개체 무결성 제약, 참조 무결성 제약 등이 있다.

■ **영역 무결성 제약(Domain Integrity Constraints) :** RDB에서 영역 무결성 제약은 테이블의 어떤 필드에 유효한 값만을 허용되는 데이터 값에 대한 제약 조건

(데이터의 길이, 데이터 값의 범위, 데이터의 타입 등)을 말한다.

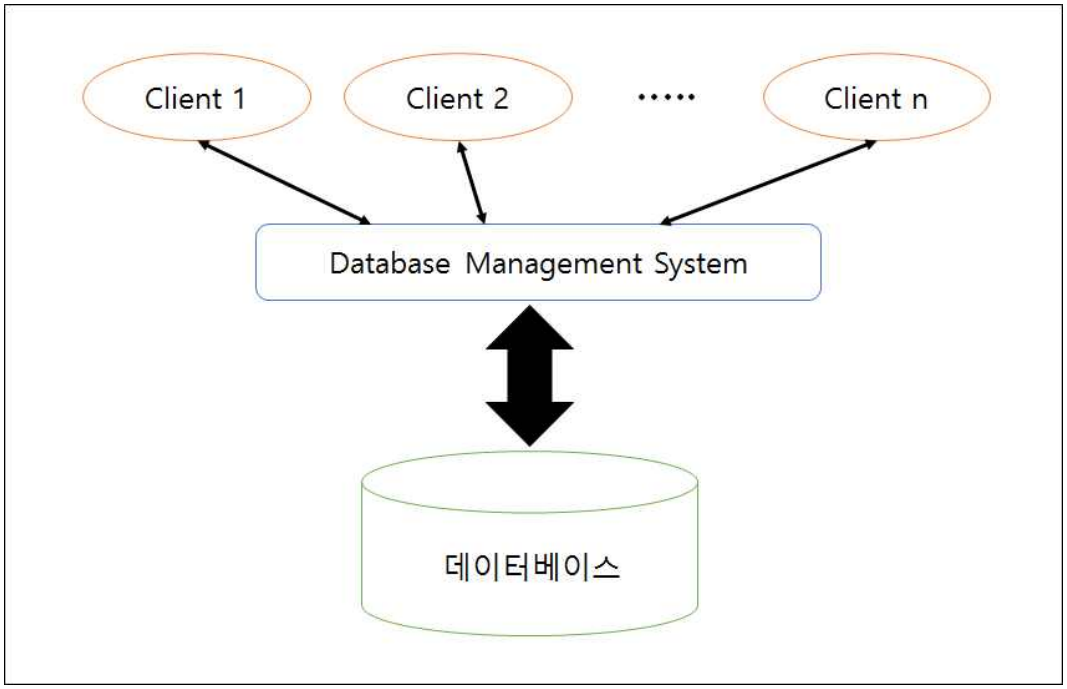
- **개체 무결성 제약(Entity Integrity Constraints) :** RDB에서 개체 무결성 제약은 테이블의 각 행을 유일한 값으로 식별 하도록 하는 제약 조건을 말한다. 기본 키 제약이 이에 속한다.
- **참조 무결성 제약(Referential Integrity Constraints) :** RDB에서 참조 무결성 제약은 참조하는 두 테이블의 관계에서 튜플 간의 일관성을 유지하기 위해서 명시하는 제약 조건을 말한다. 즉, 참조 무결성 제약은 참조하는 관계의 튜플 값이 반드시 참조되는 튜플에 존재 하여야 한다는 제약이다. 이 참조 무결성 제약은 한 관계에 있는 특정 튜플의 존재 여부가 다른 관계의 튜플의 존재 여부에 영향을 미치는 경우에 발생한다. 외래 키 제약이 이에 속한다.

2. 데이터베이스 관리 시스템

데이터베이스를 생성, 관리, 데이터로부터 사용자의 물음에 대한 대답을 추출하는 프로그램의 집합을 데이터베이스관리시스템(Database Management System, 이하 DBMS)이라 한다. DBMS은 바로 데이터를 저장하고, 데이터로부터 유용한 정보를 얻어내기 위한 효율적이면서도 편리한 방법을 사용자에게 준다.

DBMS가 데이터를 관리하기 위해서는 저장할 데이터의 구조를 정의하고, 정의된 구조에 따라 효율적으로 데이터를 저장한다. 또한 이렇게 저장된 데이터로부터 좀 더 빠르게 정보를 추출할 수 있기 위한 방법을 제공한다.

데이터베이스는 동시에 여러 사람이 사용하는 것으로 하나에 데이터베이스가 동시에 여러 장소에서 사용될 경우 한 명의 사용자가 사용할 때와는 다른 문제를 발생 시킬 수 있다. 데이터베이스를 효율적으로 사용하기 위해서는 데이터베이스에 접근하는 것을 가능하도록 하고, 발생할 수 있는 문제를 DBMS를 이용해 해결한다.



<그림 2-2> Structure of RDBMS

3. SQL

ISO/ANSI 표준 RDB의 정의와 조작을 규정한 데이터베이스 언어를 구조화된 질의어라고 부른다. 이를 SQL(Structured Query Language)이라 한다. SQL은 RDB에 대한 데이터 정의와 데이터 조작, 그리고 데이터 제어를 규정한 언어이다.

■ **데이터 정의 언어(DDL : Data Definition Language)**

데이터베이스의 논리 구조를 스키마라고 한다. RDB의 기본 구조는 테이블이며, 각 테이블에 속성을 기술하는 것이 스키마이다. 이 스키마의 정의, 변경, 삭제를 행하는 연산을 데이터 정의 연산이라고 한다. 이러한 데이터 정의 연산을 기술하기 위한 언어가 데이터 정의 언어이다. 또한 데이터베이스의 구조를 정의할 뿐 아니라 정합성 제약 등을 선언하기도 한다. 데이터 정의 언어는 다른 말로 스키마 정의 언어 또는 데이터 기술 언어라고도 한다.

데이터 구조를 정의하는 데이터 정의 언어는 'CREATE', 'DROP', 'ALTER' 등으로 구성되어 있다.

■ **데이터 조작 언어(DML : Data Manipulation Language)**

데이터 조작 언어는 데이터 조작 연산을 기술하기 위한 언어이다. 데이터 조

작 연산이란 RDB의 실제 데이터에 대한 조작 연산이며, 검색 연산과 갱신 연산이 있다.

검색 연산이란 데이터베이스로부터 사용자가 필요로 하는 정보를 뽑아내어 재구성하는 연산이다. 여기서 재구성한 데이터 구조의 스카마를 목표 스키마라 한다. 또한 정보를 뽑아내기 위한 조건을 검색 조건이라 한다.

갱신 연산이란 데이터베이스 내용의 일부를 변경하는 연산이다. 어떠한 갱신도 무제한으로 허용하면 데이터베이스 내용의 정확성이나 정합성을 보장할 수 없게 될 가능성이 있다. 따라서 몇 개의 조건을 정해 놓고 갱신한 데이터베이스가 이 조건을 유지하도록 갱신을 제한하도록 한다. 이 조건을 정합성 제약조건이라고 한다.

데이터를 조회하거나 변경하는 데이터 조작 언어는 'Select', 'Insert', 'Update', 'Delete' 등으로 구성되어 있다.

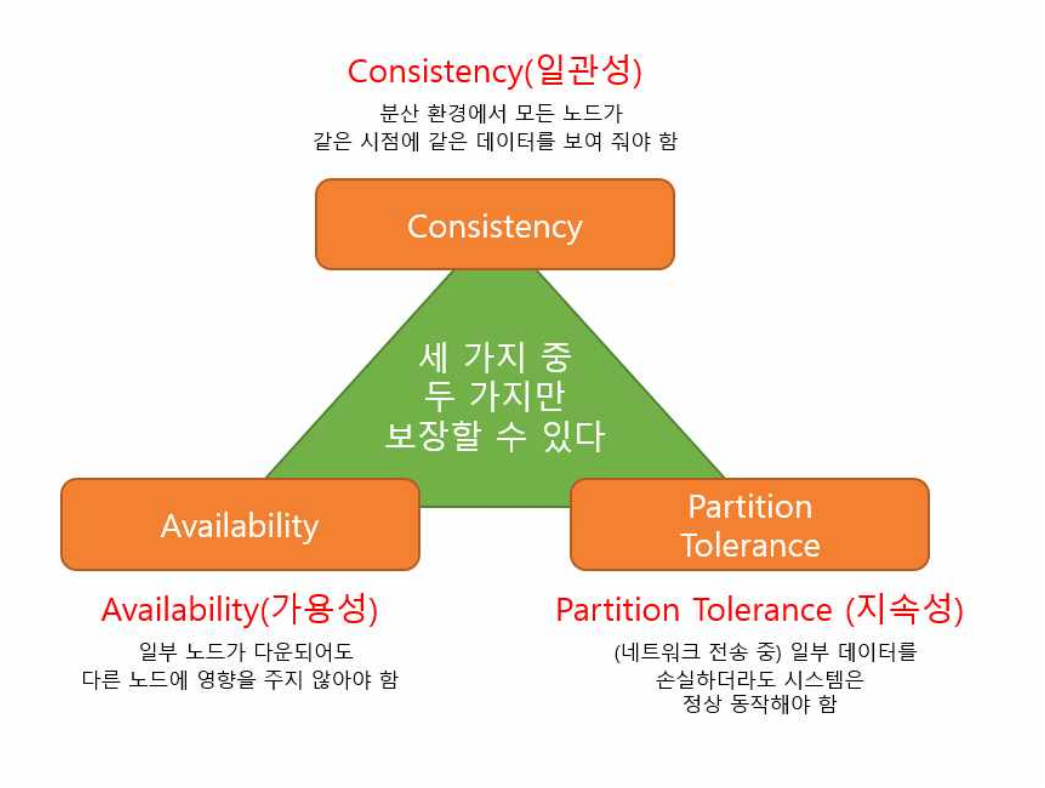
■ **데이터 제어 언어(DCL : Data Control Language)**

데이터 제어 언어는 데이터를 제어하기 위한 언어이다. 데이터의 보안, 무결성, 회복, 병행 수행제어 등을 정의하는데 사용한다. 데이터를 제어하는 데이터 제어 언어는 'Commit', 'Grant', 'Revoke' 등으로 구성되어 있다.

제2절 NoSQL

기존의 데이터 저장 구조는 디스크에 저장된 전체 데이터베이스에 액세스하기에 처리 속도가 늦고 관리가 어려웠다. 또한 저장할 데이터양의 증가로 읽기/쓰기에서 RDBMS의 제약이 생기고, 수평 확장에도 한계가 있었다. 더불어 웹 서비스 구조가 변화하여 저장할 데이터의 형태가 다양해졌으며, 사용자의 데이터 요구 또한 다변화했다. 특히 <그림 2-3>의 CAP(Consistency, Availability, Partition Tolerance) 이론 중 일관성(Consistency)과 가용성(Availability)을 만족시키려고 만든 RDBMS로는 해결 하지 못하는 여러 문제에 부딪혔다. 따라서 새로운 저장 기술이 필요하게 되었고 그에 적합한 대표적인 기술이 바로 NoSQL(Not Only SQL)이다.

NoSQL은 테이블 스키마(Table Schema)가 고정되지 않고, 테이블 간 JOIN 연산을 지원하지 않으며, 수평적 확장이 용이하다는 특징을 가진다. RDBMS의 경우, 일관성과 가용성에 중점을 두고 있는 반면, NoSQL 기술은 지속성에 중점을 두고 일관성과 유효성은 보장하지 않는다. 이것은 일관성, 유효성, 지속성 중 2가지만 보장이 가능하다는 분산 데이터베이스 시스템 분야의 CAP 이론에 따른 것이다. 따라서 대규모의 유연한 데이터 처리를 위해서는 NoSQL 기술이 적합하지만, 안정성이 중요한 시스템에서는 오랫동안 검증된 RDBMS를 채택할 필요가 있다.



<그림 2-3> CAP Theory

1. NoSQL 특징

NoSQL은 제품 종류에 따라서 지원하는 기능이 다르긴 하나, 공통적으로 정렬이나 JOIN 연산, GROUP BY 연산 등은 지원하지 않아 필요할 때마다 추가로 구현하여 사용해야 한다는 단점이 있다. 그래서 이런 단점을 보완한 NoSQL 제품과 다양한 오픈 소스 제품이 등장했는데, 개발 목적에 맞는 제품을 선택하려면 각 제품의 특징부터 알아야 한다. 이런 NoSQL의 기술적 특성을 정리하면 <표 2-1>과 같다.

<표 2-1> Technical Characteristics of NoSQL

특성	내용
無 스키마	• 데이터를 모델링하는 고정된 데이터 스키마 없이 키 값을 이용하여 다양한 형태의 데이터 저장과 접근 가능 • 데이터 저장 방식은 크게 열, 값, 문서, 그래프의 네 가지를 기반으로 구분
탄력성	• 시스템 일부에 장애가 발생해도 클라이언트가 시스템에 접근 가능 • 응용 시스템의 다운 타임이 없도록 하는 동시에 대용량 데이터가 생성과 갱신됨
질의 기능	• 수십 대에서 수천 대 규모로 구성된 시스템에서도 데이터의 특성에 맞게 효율적으로 데이터를 검색, 처리할 수 있는 질의 언어, 관련 처리 기술, API 제공
캐싱(caching)	• 대규모 질의에도 고성능 응답 속도를 제공할 수 있는 메모리 기반 캐싱 기술 적용 • 개발과 운영에도 투명하고 일관되게 적용할 수 있는 구조임.

NoSQL은 데이터 모델에 따라 크게 세 가지로 구분된다. 첫 번째는 <키, 값> 저장 구조, 두 번째는 문서 저장 구조, 세 번째는 열 기반 저장 구조이다. <키, 값> 저장 구조는 가장 간단한 데이터 모델이지만, 범위 질의에 약하고 응용 프로그램 모델링이 복잡해진다는 단점이 있다. 문서 저장 구조는 RDBMS의 고정된 스키마와 달리 새로운 스키마 구조이며, 레코드 관계를 설명할 수 있는 기존 RDBMS와 아주 비슷한 개념의 NoSQL 분류이다.

마지막으로 열 기반 저장 구조는 연관된 데이터 위주로 읽어 오는 데 유리한 구조로, 범위 질의에 적합하고 압축 효율이 높다는 장점이 있다. 이런 NoSQL의 분류를 요약하면 다음과 같다.

<표 2-2> NoSQL Data Model Characteristics

데이터 모델	설명
<키, 값> 저장구조	• 가장 간단한 데이터 모델 • 범위 질의는 사용이 어려움 (DB에서 지원하면 사용 가능) • 응용 프로그램 모델링이 복잡함
열 기반 저장 구조	• 연관된 데이터 위주로 읽는 데 유리한 구조 • 하나의 레코드를 변경하려면 여러 곳을 수정해야 함 • 돌일 도메인의 열 값이 연속되므로 압축 효율이 좋음 • 범위 질의에 유리
문서 저장 구조	• 문서에는 다른 스키마가 있음 • 레코드 간의 관계 설명이 가능 • 개념적으로 RDBMS와 비슷

2. RDBMS와 NoSQL 비교

다음 표는 기존 RDBMS와 NoSQL을 CAP 이론에 맞게 비교한 것이다.

<표 2-3> Comparison of RDBMS and NoSQL based on CAP Theory

구분	설명	적용 예
RDBMS	일관성(C)과 가용성(A)을 선택	트랜잭션 ACID의 보장 (금융 서비스)
NoSQL	일관성(C)이나 가용성(A) 중 하나를 포기하고, 지속성(P)을 보장	• C+P형: 대용량 분산 파일 시스템(성능 보장) • A+P형: 비동기식 서비스(아마존, 트위터 등)

<표 2-4> Characteristics, pros and cons of RDBMS and NoSQL

구분	RDBMS	NoSQL
장·단점	• 데이터 무결성, 정확성 보장 • 정규화된 테이블과 소규모 트랜잭션이 있음 • 확장성에 한계가 있음 • 클라우드 분산 환경에 부적합	• 데이터 무결성과 정확성 보장하지 않음 • 웹 환경의 다양한 정보를 검색·저장 가능 • 하드웨어 확장에 유연한 대처가 가능 • RDBMS 대비 저렴한 비용으로 분산처리와 병렬처리 가능 • 비정형 데이터 구조 설계로 비용감소
특성	• UPDATE, DELETE, JOIN 연산 기능 • ACID 트랜잭션이 있음 • 고정 스키마가 있음	• 수정, 삭제를 사용하지 않음(입력으로 대체) • 강한 일관성은 불필요 • 노드의 추가 및 삭제, 데이터 분산에 유연

< NoSQL 데이터 모델 >

■ **<키, 값> 저장구조**

이 아이디어는 특정 데이터에 대한 유일 키나 포인터가 있다면, Hash Table을 사용하자는 것이다. <키, 값> 저장구조 모델은 매우 구현하기 쉽고, 간단하다. 그러나 값의 일부분을 읽거나 업데이트 한다면 매우 비효율적이다. Cassandra가 이에 속한다.

■ **열 기반 저장 구조**

이 기술은 여러 서버에 분산된 수많은 데이터를 저장, 처리하기 위해 만들어졌다. 여전히 키를 사용하지만, 키는 여러 개의 컬럼을 가리키고 있다. 컬럼은 컬럼 패밀리에 따라 정렬되어 진다. HBase가 이에 속한다.

■ **문서 저장구조**

이 기술은 기본적으로 <키, 값> 저장구조와 유사하다. 이 모델에서 문서는 많은 <키, 값> 콜렉션들의 콜렉션이다. 반 정형화된 문서들이 JSON(JavaScript Object Notation)과 같은 포맷으로 저장되어진다. 문서 데이터베이스는 기본적으로 <키, 값> 저장 구조의 차세대 버전이다. MongoDB가 이에 속한다.

3. NoSQL의 종류

3.1. Cassandra

Cassandra는 <키, 값> 구조의 DBMS로, DynamoDB를 설계한 사람 중 한 명인 Avinash Lakshman과 페이스북의 Proshant Malik이 최초로 개발했다. 페이스북에 적용하여 사용하다가 2008년 구글을 이용하여 오픈 소스로 배포했다. 많은 사용자의 요구를 반영하여 다양한 기능을 포함하는 형태로 진화했으며, 2009년 3월 아파치의 인큐베이터 프로젝트가 된 지 불과 1년 만인 2010년 2월 아파치 최상위 프로젝트로 선택되었다.

Cassandra는 아마존 DynamoDB의 해시 알고리즘을 이용한 <키, 값> 저장 구조와 계층적 열 구조를 논리 구조의 배경으로 한다.

<그림 2-4> Data Storage Structure of Cassandra

Cassandra는 페이스북의 인박스 환경 때문에 개발한 만큼 방대한 양의 데이

터 저장과 처리, 정렬에 활용되는 속성 제한, 빈번한 데이터 변경에 사용하기 가장 좋은 환경이라고 볼 수 있다.

< Cassandra 주요 특징 >

- 토큰링 배경의 키 구간이 설정되어 있어 서버의 추가 및 제거만으로도 전체 저장 공간의 유연한 확장 및 축소 가능
- 다른 서버에 데이터 복제본을 구성하여 특정 노드에 장애가 발생해도 서비스에 영향을 주지 않고, 데이터가 유실되지 않음
- 수정/추가/삭제할 때 실제 스토리지 구조에 적용하기 전에 먼저 CommitLog에 변경 사항을 기록하므로 성능이 빠름
- 1차 인덱스는 열 집합의 열 이름, 2차 인덱스는 열의 값을 기반으로 함
- Thrift를 데이터 전송 프로토콜로 사용. 언어에 종속 없이모든 환경에서 이용 가능
- 물리 파일 저장 구조로 SSTable 사용. 물리 파일을 저장하는 구조 자체가 1차 정렬 조건에 맞추어 사전 정렬된 형태 유지

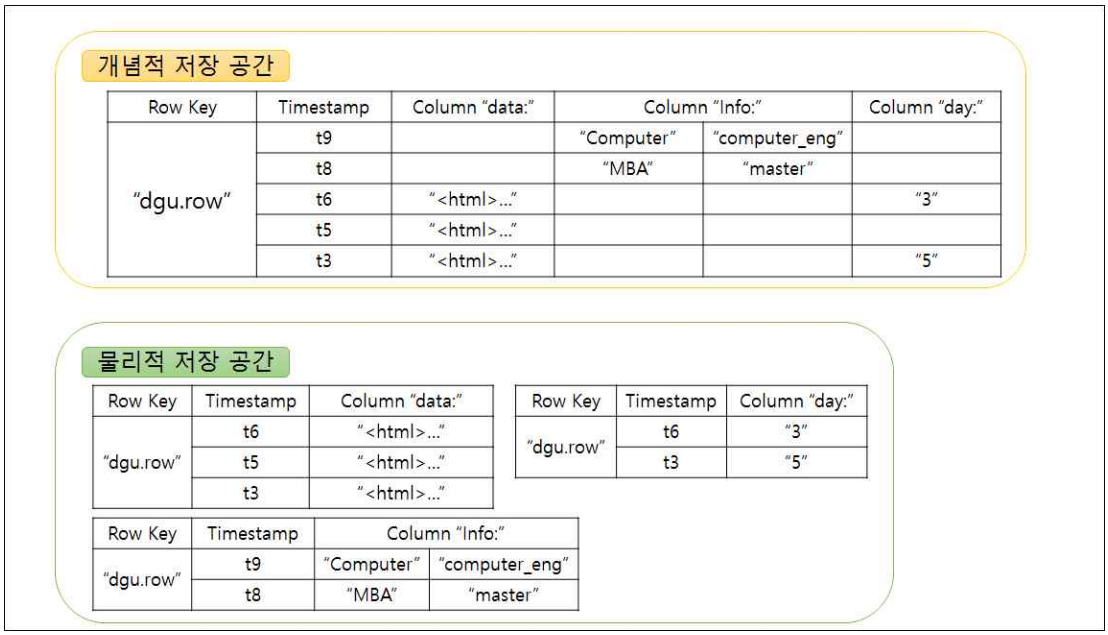
< Cassandra 구성 데이터 모델 >

- Column : 컬럼은 컬럼 이름과 값으로 이루어진 데이터 구조체
- Column Family : 컬럼 패밀리는 컬럼들의 집합으로 하나의 Row를 식별하기 위한 키를 갖고, 하나의 키에 여러 개의 컬럼이 달려 있는 형태임
- KeySpace : 논리적으로 컬럼 패밀리를 묶어주는 개념. 데이터구조나 관계에서 특별한 영향을 주지 않음
- Super Column, Super Column Family : 컬럼에서 컬럼의 값은 'String' 이나 'Integer' 와 같은 원시적 형태뿐만 아니라 컬럼 자체가 다시 들어갈 수 있음
- Thrift : 페이스북이 개발한 가변적 규모의 이종 언어 서비스 개발을 위한 S/W Framework임. 다양한 언어를 사용하여 개발한 S/W를 쉽게 결합하기 위해 Thrift 사용. Apache에서 제공하는 Thrift는 다양한 플랫폼 간에 매우 편리하게 사용할 수 있는 통합 RPC(Remote Procedure Call) 환경 제공. 스크립트 작성시 다양한 언어에서 사용될 수 있는 코드를 자동으로 생성

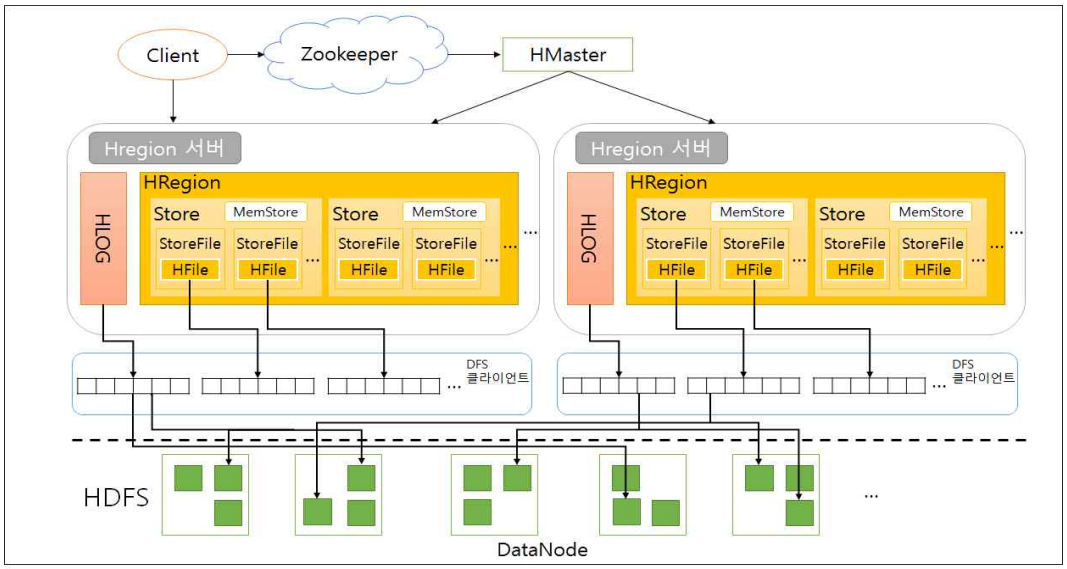
3.2. HBase

HBase는 행, 열, 그룹, 열 이름, 타임스탬프를 이용한 테이블 구조로 되어 있다. HDFS 위에 설치되며, Zookeeper를 노드 관리에 사용한다. 대표적인 사용 예로 웹 테이블(수집된 웹 페이지와 웹 페이지 URL을 키로 하는 속성 테이블)이 있으며, Adobe, Twitter, Yahoo 등에서 사용한다. 읽기와 수정을 즉시 실행되며, Map-Reduce 연산은 일괄 처리된다. 또한 각 프로세스는 자신의 레코드를 비동기적으로 업데이트 할 수 있다. 영역 서버 간의 시스템 대체 작동과 불량 클러스터 복구 기능, Get, Put, Scan, Delete의 네 가지 동작을 지원한다. 데이터 모델은 열 기반 저장 구조로 구성되어 있다.

<그림 2-5>은 HBase의 개념적, 물리적 저장 공간을 비교한 것이고, <그림 2-6>은 HBase의 구조이다.



<그림 2-5> Compare Physical and Conceptual Storage of HBase



<그림 2-6> Structure of HBase

클라이언트는 사전 정의된 루트 영역 테이블이 있는 영역 서버를 관리하는 마스터 서버에 접속한다. 그런 다음 User 테이블과 영역 서버 매핑 정보가 있는 메타-데이터 영역 테이블을 관리하는 두 번째 영역 서버에 접속한다. 두 번째 서버는 키 범위와 영역 서버를 Mapping 하는 세 번째 서버의 종료 포인트에 응답한다. 세 번째 서버는 사용자 영역을 조회하고 클라이언트가 찾고 있는 데이터를 네 번째 영역 서버에 전달한다. 인메모리 데이터 저장을 메모리 캐시에서 구현하여 찾는 데이터가 없으면 HDFS에서 조회한다. 디스크의 데이터를 저장한 하둡 데이터노드는 서버에 거주하는 로그를 포함하는 HDFS 파일로 구현한다.

< HBase 데이터 구성 >

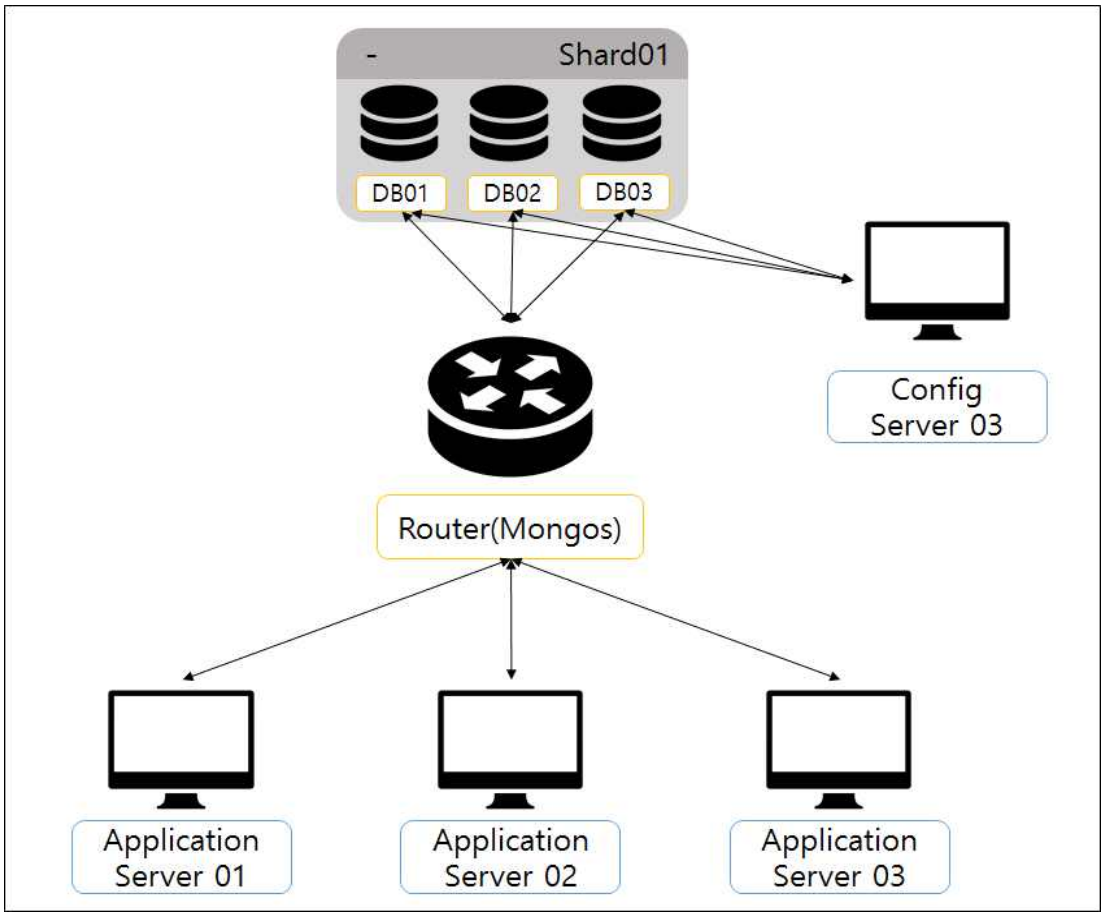
- Table : Row들의 집합으로 스키마 정의 시 Column Family만 정의
- Row Key : 임의의 Byte열로 사전 순으로 내림차순 정렬됨. 빈 Byte문자열은 테이블의 시작과 끝을 의미. 문자열, 정수, 바이너리, 직렬화 된 데이터 구조까지 어떤 것도 Row Key가 될 수 있음
- Column Family : Column들의 그룹으로 모든 Column Family의 요소는 같은 접두사 사용. Column Family의 접두사는 반드시 표시할 수 있는 문자로 구성. 테이블 스키마에서 정의의 한 부분을 먼저 지정해야함. 모든 Column Family의 요소는 물리적으로 파일시스템에서 함께 저장됨
- Cell : Row Key, Column, Version 이 명시된 튜플. Table Cell은 버전관리 가능

3.3. MongoDB

MongoDB는 신뢰성과 확장성에 기반한 문서 지향 데이터베이스이다. 방대한 양의 데이터에서 낮은 관리 비용과 사용 편의성을 목표로 하는 MongoDB는 10gen이 오픈 소스로 개발한 것으로, 상업적인 지원이 가능하다. MongoDB에서 저장의 최소 단위는 문서로, 이는 RDBMS의 행과 비슷하다. 각 문서들은 RDBMS의 테이블과 비슷한 컬렉션이라는 곳에 수집하며, 각 컬렉션은 데이터베이스에서 관리한다. MongoDB는 자동화된 샤딩(Sharding)으로 기능 손실이 없는 수평적 확장이 가능하고, 기존 RDBMS의 범위 질의, 보조 인덱스, 정렬 등 연산과 Map-Reduce 등 집계 연산을 함께 지원한다. 데이터는 JSON 형태로 저장하며, C++로 작성한다.

MongoDB에서는 문서를 컬렉션으로 모으며, 스키마가 필요 없다. 그리고 이론적으로 컬렉션 하나에 포함된 문서는 구조가 서로 달라도 되나, 실제로는 구조가 균일하다. 예를 들어, '성적'이 라는 컬렉션에는 '학번, 이름, 과목명, 점수'와 같은 필드가 있다. 그리고 MongoDB의 질의는 JavaScript로 작성하고, 문서 기반의 질의를 지원한다. 이는 셸로 실시간 접근이 가능하며, 다양한 프로그래밍 언어를 지원한다.

MongoDB는 자동-샤딩(Auto-Sharding)을 이용한 분산 확장이 가능하다. 샤딩은 데이터를 분할하여 다른 서버에 나누어 저장하는 과정을 말하며, 데이터를 여러 서버로 분할 저장함으로써 더 많은 데이터를 관리하고 처리할 수 있다. 그러나 대부분의 DBMS는 샤딩을 응용 프로그램 단에서 구성하기에 어려움이 있는데, MongoDB는 자동-샤딩을 제공하여 이를 해결한다. 예를 들어, 방대한 양의 데이터 때문에 시스템의 속도가 떨어질 때 자동-샤딩을 이용하면 여러 대의 MongoDB에 데이터를 분산하여 저장할 수 있다. 그리고 서버의 용량이 부족할 때 새로운 샤드(Shard) 노드를 추가하면 자동으로 재분산을 수행한다. 이것은 사용자가 분산 구조를 고려하지 않고도 확장이 가능하도록 하는 기능이다.



<그림 2-7> Expansion Method Using Auto-Sharding of MongoDB

제3장 빅데이터 처리 방법론

제1절 NoSQL 기반 처리 방법론

NoSQL의 경우 분산처리에 적합하게 제작이 되었지만, 분산처리 환경을 구축하기 위해서는 많은 장비와 자원이 소모가 된다. 단일 노드에서 동작하는 NoSQL 기반의 침입 탐지 시스템은 분산처리 환경에 비하여 성능이 떨어지겠지만, 개인 컴퓨터에서 동작하는 침입 탐지 시스템으로 적합하게 사용될 것으로 보인다.

1. Cassandra

1.1. 처리 방법론

처음으로 제시하는 시스템은 단일 노드에서 Cassandra를 이용한 침입 탐지 시스템이다. Cassandra는 <키, 값> 구조의 데이터베이스로 방대한 양의 데이터 저장과 처리, 정렬에 좋은 환경을 가지고 있다. Cassandra는 CQL 이라는 언어를 사용하여 데이터베이스의 데이터를 저장하고 처리한다. CQL은 다른 말로 SQL for Cassandra라고 하는데, SQL문과 유사한 방식의 형식을 가지고 있기 때문에, SQL 사용자들이 단시간에 익숙해질 수 있는 장점이 있다. 또한 MySQL의 행, 열 구조와 Cassandra의 <키, 값> 구조가 유사하기 때문에 시스템 변경 시 기존 시스템에 익숙해져 있던 관리자들도 큰 부담 없이 시스템을 옮겨 갈 수 있을 것이다.

Cassandra는 데이터 전송 프로토콜로 Thrift를 사용하여 언어에 의존 없이 모든 환경에서 사용이 가능하므로, Cassandra기반의 침입 탐지 시스템 구축 시 다양한 언어의 지원이 가능해질 것으로 보인다. Cassandra는 데이터 저장 시 CommitLog에 변경 사항을 기록함으로써 데이터의 수정, 추가, 삭제가 빠르게 진행이 되어 데이터가 빈번히 변경되는 환경에서 가장 좋다고 볼 수

있지만, 보안 이벤트의 특성 상 데이터의 변조/수정이 일어나게 되면 침입 탐지를 할 수 없을 수 있기 때문에 이러한 특징은 보안 이벤트 분석 시스템 구축 시 시스템 구성 요소에 맞게 선택할 필요가 있다.

Cassandra는 'Partial Word Search'를 자체적으로 지원하지 않는다. 따라서, 침입을 탐지하기 위해서는 Cassandra 자체적으로 불가하기 때문에 Apache Solr와 접목하여 'Solandra' 라는 시스템을 만든다. Cassandra는 데이터베이스의 역할을 Solr는 검색플랫폼의 역할을 수행하여 Cassandra 자체 성능보다 빠르게 데이터를 검색할 수 있게 된다.

Cassandra의 성능을 검증하기 위해서 단일 노드에 Cassandra 데이터베이스를 구축하고 Snort Signatrue 기반의 데이터를 저장한 후 저장된 데이터를 분석함으로써 Cassandra의 단일 노드 성능을 검증하는 방법을 제시한다.

1.2. 빅데이터 처리 프로토타입 시스템 설계

처음으로 설계하는 시스템은 <키,값> 저장구조의 대표적인 제품인 Cassandra에 스노트 시그니쳐를 접목한 시스템이다. 단일 노드위에 Cassandra를 설치하고 데이터베이스에 스노트 시그니쳐 더미 데이터를 생성하여 저장한다. 저장된 스노트 시그니쳐 데이터를 이용하여 보안 이벤트 데이터를 처리한다. 처리 과정은 <그림 3-1>과 같다.

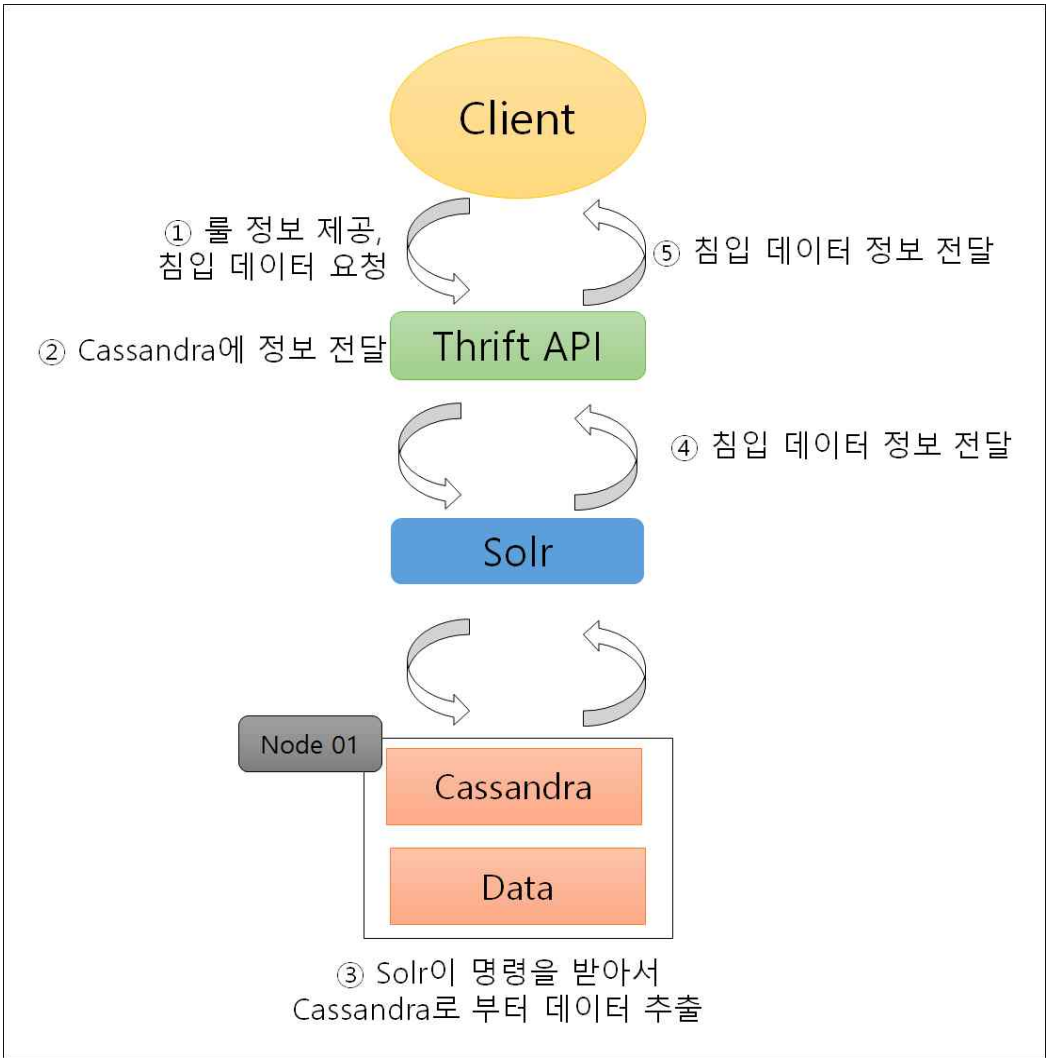
① 관리자는 클라이언트를 이용하여 Cassandra 서버로 침입 데이터에 대한 데이터를 요청하며, 이와 함께 침입 데이터를 판단할 수 있는 Role 정보를 같이 전송

② Thrift API는 클라이언트로부터 받은 정보를 Cassandra가 이해할 수 있는 언어로 변환하여 Cassandra에 쿼리 전달

③ 검색 플랫폼인 Solr는 Thrift API가 전달한 Role 정보를 가지고 Cassandra 데이터베이스에 저장되어 있는 스노트 시그니쳐 데이터 중 매칭 되는 데이터를 추출하여 클라이언트로 데이터 전송

④ Cassandra 데이터베이스에서 추출된 침입 데이터를 Thrift API가 전달받아 클라이언트에서 이해할 수 있는 언어로 변환하여 클라이언트에 데이터 전달

⑤ 클라이언트는 Thrift API로부터 Cassandra 데이터베이스가 전달한 침입데이터를 전달받으며, 관리자에게 전달 받은 데이터가 보안상 위협 요소가 있음을 통지

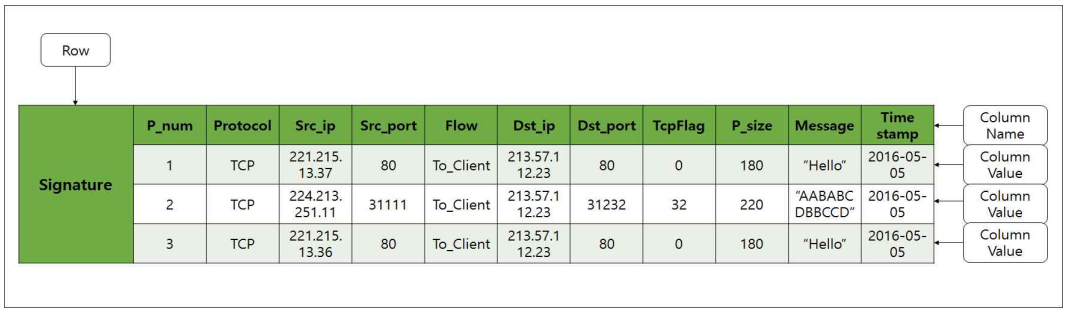


<그림 3-1> Standalone Cassandra Operating Structure

위와 같은 방식으로 단일 노드에서 동작하는 Cassandra 기반의 보안 이벤트 처리 프로토타입 시스템을 구축할 수 있다.

다음으로는 Cassandra 데이터베이스 구축을 하고 데이터베이스에 저장을 할 스노트 시그니쳐 더미 데이터의 생성 방법이다.

Cassandra는 <키, 값> 저장 구조이기 때문에 한 키에 하나의 값이 들어가는 방식이다. <키, 값> 구조에 적합하게 스노트 시그니쳐 데이터를 생성하여 저장을 하여야 한다.



<그림 3-2> Snort Signature Data Structure in Cassandra

<그림 3-2>는 Cassandra에서의 스노트 시그니쳐 데이터 구조이며, <표 3-1>은 각 필드에 대한 설명이다.

<표 3-1> Cassandra의 스노트 시그니쳐 데이터 구조

구분	설명
P_num	• 저장된 패킷의 번호. 순차적으로 값이 증가하며, 관리자가 패킷의 다른 정보를 얻기 위해 구분하는 번호임
Protocol	• TCP/IP 프로토콜 정보. 'TCP', 'UDP', 'ICMP' 등의 정보 포함
Src_ip	• 공격자 IP 주소. IPv4 형식의 주소. Role에서 특정 주소를 가리키지 않는 any를 Role로 적어 놓은 경우 모든 주소가 매칭됨
Src_port	• 공격자의 포트 번호. 특정 포트를 가리키지 않는 any를 Role로 적어 놓는 경우 모든 포트가 매칭됨. 특정 포트나 포트 범위가 Role에 적혀 있을 때는 포트 번호를 비교 매칭
Flow	• 패킷의 이동 방향. To_Client 혹은 To_Server의 값이 저장
Dst_ip	• 목적지의 IP 주소. IPv4 형식의 주소. Role에서 특정 주소를 가리키지 않는 any를 Role로 적어 놓은 경우 모든 주소가 매칭됨
Dst_port	• 목적지의 포트 번호. 특정 포트를 가리키지 않는 any를 Role로 적어 놓는 경우 모든 포트가 매칭. 특정 포트나 포트 범위가 Role에 적혀 있을 때는 포트 번호를 비교 매칭
TcpFlag	• TCP/IP의 Flag 비트 값. TCP/IP의 Flag는 URG(32), ACK(16), PSH(8), RST(4), SYN(2), FIN(1) 6개로 구성
P_size	• 패킷의 크기. 패킷의 크기를 분석하여 오버플로우 공격 예방 가능
Message	• 패킷에 실린 데이터의 내용. 침입 데이터 분석에 있어서 가장 중요한 열이라고 할 수 있으며, 해당 열에 저장된 데이터를 분석하여 Role과 비교 후 침입데이터 유무 파악
Timestamp	• 패킷이 들어온 시간

위와 같은 구조로 스노트 시그니쳐기반의 패킷 데이터를 저장하고, 저장된 데이터 중에 침입으로 간주되는 데이터를 추출한다. Client에서 데이터 추출 명령이 떨어지고, 명령이 데이터베이스에 도달하여 명령이 수행되고 도출된 결과물이 클라이언트까지 도달하는 시간을 측정한다.

2. HBase

2.1. 처리 방법론

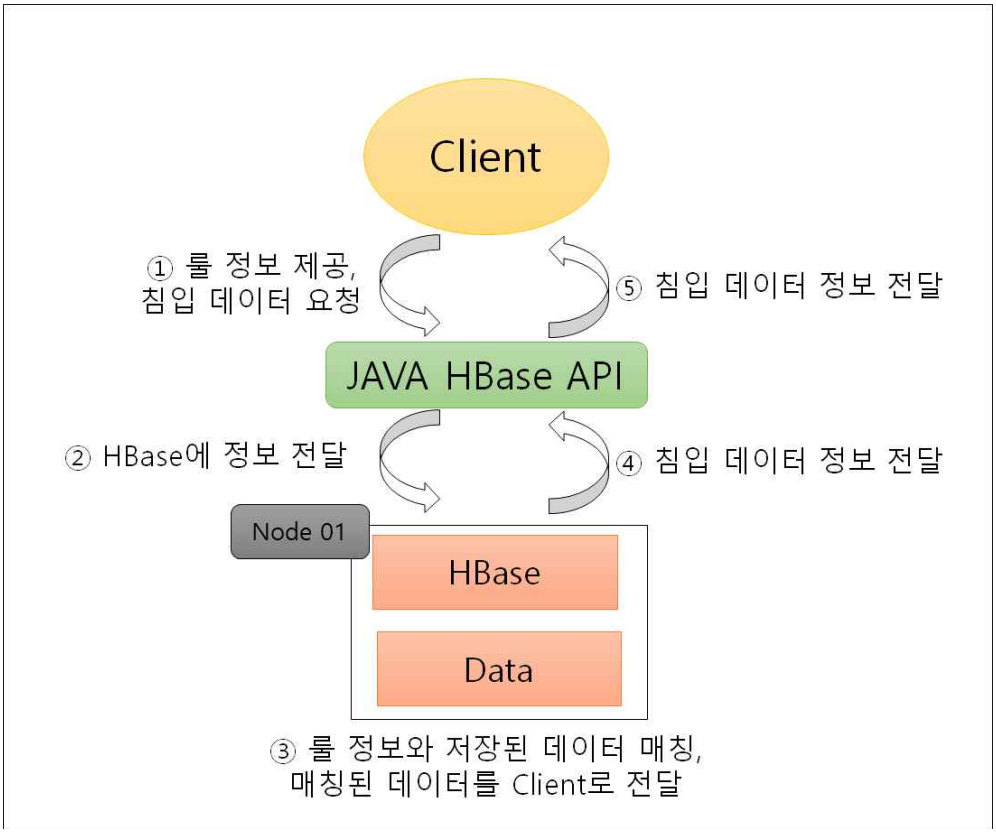
다음 제시하는 시스템은 단일 노드에서 HBase를 이용한 침입 탐지 시스템이다. HBase는 열 기반 저장 구조의 대표적인 제품이다. HBase는 하둡 위에서 동작하는 데이터베이스이기 때문에 하둡의 장점을 모두 이용할 수 있다. 하지만 HBase를 단일 노드에서 동작시키는 경우 HDFS를 사용하지 않고 로컬 시스템에 데이터를 저장시킨다. 따라서 복제되는 데이터가 없기 때문에 분산처리 시스템에 비해 스토리지가 감소 한다는 장점이 있다. HBase는 열 기반 저장 구조이기 때문에 Column 데이터의 변화에 바로바로 대응 할 수 있다. 스노트 시그니쳐의 옵션에는 다양한 종류의 값이 들어갈 수 있는데, MySQL에서 모든 옵션의 경우를 예비해서 테이블의 값을 크게 잡고 비어있는 값을 넣게 되면 데이터의 양도 증가하고 성능도 감소하게 된다. HBase의 경우에는 Row Key와 Column Family의 관계로 값을 찾기 때문에 스노트 시그니쳐의 옵션의 변화에도 적극적으로 대응이 가능하다.

HBase의 성능을 검증하기 위해서 단일 노드에 HBase 데이터베이스를 구축하여 HDFS를 사용하지 않고 로컬 시스템에 데이터를 저장하는 시스템을 구축한다. 그리고 스노트 시그니쳐 기반의 데이터를 저장한 후 저장된 데이터를 분석함으로써 HBase의 단일 노드 성능을 검증하는 방법을 제시한다.

2.2. 빅데이터 처리 프로토타입 시스템 설계

단일 노드위에 HBase를 설치하고 데이터베이스에 스노트 시그니쳐 더미 데이터를 생성하여 저장한다. 저장된 스노트 시그니쳐데이터를 이용하여 보안

이벤트 데이터를 처리한다. 처리 과정은 <그림 3-3>과 같다.



<그림 3-3> Standalone HBase Operating Structure

① 관리자는 클라이언트를 이용하여 HBase 서버로 침입 데이터에 대한 데이터를 요청하며, 이와 함께 침입 데이터를 판단할 수 있는 Role 정보를 같이 전송

② JAVA HBase API는 클라이언트로부터 받은 정보를 HBase가 이해할 수 있는 언어로 변환하여 HBase에 쿼리 전달

③ HBase 데이터베이스는 클라이언트가 JAVA HBase API를 통해 전달한 Role 정보를 가지고 저장되어 있는 스노트 시그니쳐 데이터 중 매칭 되는 데이터를 추출하여 클라이언트로 데이터 전송

④ HBase 데이터베이스에서 추출된 침입 데이터를 JAVA HBase API가 전달받아 클라이언트에서 이해할 수 있는 언어로 변환하여 클라이언트에 데이터 전달

⑤ 클라이언트는 JAVA HBase API로부터 HBase 데이터베이스가 전달한 침입 데이터를 전달 받으며, 관리자에게 전달 받은 데이터가 보안상 위협 요소가 있음을 통지

위와 같은 방식으로 단일 노드에서 동작하는 HBase 기반의 보안 이벤트 처리 프로토타입 시스템을 구축할 수 있다.

다음으로는 HBase 데이터베이스 구축을 하고 데이터베이스에 저장을 할 스노트 시그니쳐 더미 데이터의 생성 방법이다.

HBase는 열 기반 저장 구조이기 때문에, 연관된 데이터 위주로 읽는데 유리한 구조이다. RDB로 쓰여진 Snort 데이터베이스 구조는 연관성이 높기 때문에, RDB에서 HBase로 쉽게 이동을 할 수 있다.

Row key	Time stamp	Column "data:"									
		Protocol	Src_ip	Src_port	Flow	Dst_ip	Dst_port	TcpFlag	P_size	Message	Time stamp
1	T1	TCP	221.215.13 .37	80	To_Client	213.57.11 2.23	80	0	180	"Hello"	2016-05-05
2	T2	TCP	224.213.25 1.11	31111	To_Client	213.57.11 2.23	31232	32	220	"AABABC DBBCCD"	2016-05-05
3	T3	TCP	221.215.13 .36	80	To_Client	213.57.11 2.23	80	0	180	"Hello"	2016-05-05

<그림 3-4> Snort Signature Data Structure in HBase

<그림 3-4>는 HBase에서의 스노트 시그니쳐 데이터 구조이며, <표 3-2>는 이에 대한 세부 설명이다.

<표 3-2> HBase의 스노트 시그니쳐 데이터 구조

구분	설명
Row key	• 저장된 패킷 번호
Timestamp	• 데이터가 생성된 시간
data:Protocol	• TCP/IP 프로토콜 정보
data:Src_ip	• 공격자의 주소
data:Src_port	• 공격자의 포트
data:Flow	• 패킷의 이동 방향
data:Dst_ip	• 목적지의 주소
data:Dst_port	• 목적지의 포트
data:TcpFlag	• TCP/IP의 Flag 비트 값
data:P_size	• 패킷의 크기를
data:Message	• 패킷에 실린 데이터 내용을 저장
data:Timestamp	• 패킷이 들어온 시간

위와 같은 구조로 Snort Signataure 기반의 패킷 데이터를 저장하고, 저장된 데이터 중에 침입으로 간주되는 데이터를 추출한다. 클라이언트에서 데이터 추출 명령이 떨어지고, 명령이 데이터베이스에 도달하여 명령이 수행되고 도출된 결과물이 클라이언트까지 도달하는 시간을 측정한다.

3. MongoDB

3.1. 처리 방법론

MongoDB는 문서 지향 데이터베이스의 대표적인 제품이다. 대용량의 데이터를 낮은 관리 비용과 편리한 사용성이 장점인 제품이다. MongoDB는 데이터를 문서 형식으로 저장하기 때문에 보안이벤트 하나하나를 문서화 시킬 수 있다. 문서는 MySQL에서 행과 비슷하기 때문에, MySQL에서 MongoDB로 이식성이 좋다. MongoDB는 데이터가 JSON 형태로 저장되기 때문에 다양한 프로그래밍 언어를 지원하여 개발 시 다른 언어를 배울 필요가 적다는 것과 구축이 쉽고 사용하기 편한다는 장점이 있지만, MongoDB를 한 대의 서버에 단일로 설치했을 때, 장애가 발생할 경우 다시 시스템을 가동하기 전까지 스스로 복구가 어렵다는 단점이 있다.

MongoDB의 성능을 검증하기 위해서 단일 노드에 MongoDB 데이터베이스를 구축하고, 스노트 시그니쳐 기반의 데이터를 저장한 후 분석하여 단일 노드에서 동작하는 MongoDB의 성능을 검증하는 방법을 제시한다.

3.2. 빅데이터 처리 프로토타입 시스템 설계

다음 시스템은 문서 기반 저장구조의 대표적인 제품인 MongoDB에 스노트 시그니쳐를 접목한 시스템이다. 단일 노드위에 MongoDB를 설치하고 데이터베이스에 스노트 시그니쳐 더미 데이터를 생성하여 저장한다. 저장된 스노트 시그니쳐 데이터를 이용하여 보안 이벤트 데이터를 처리한다. 처리 과정은 <그림 3-5>와 같다.

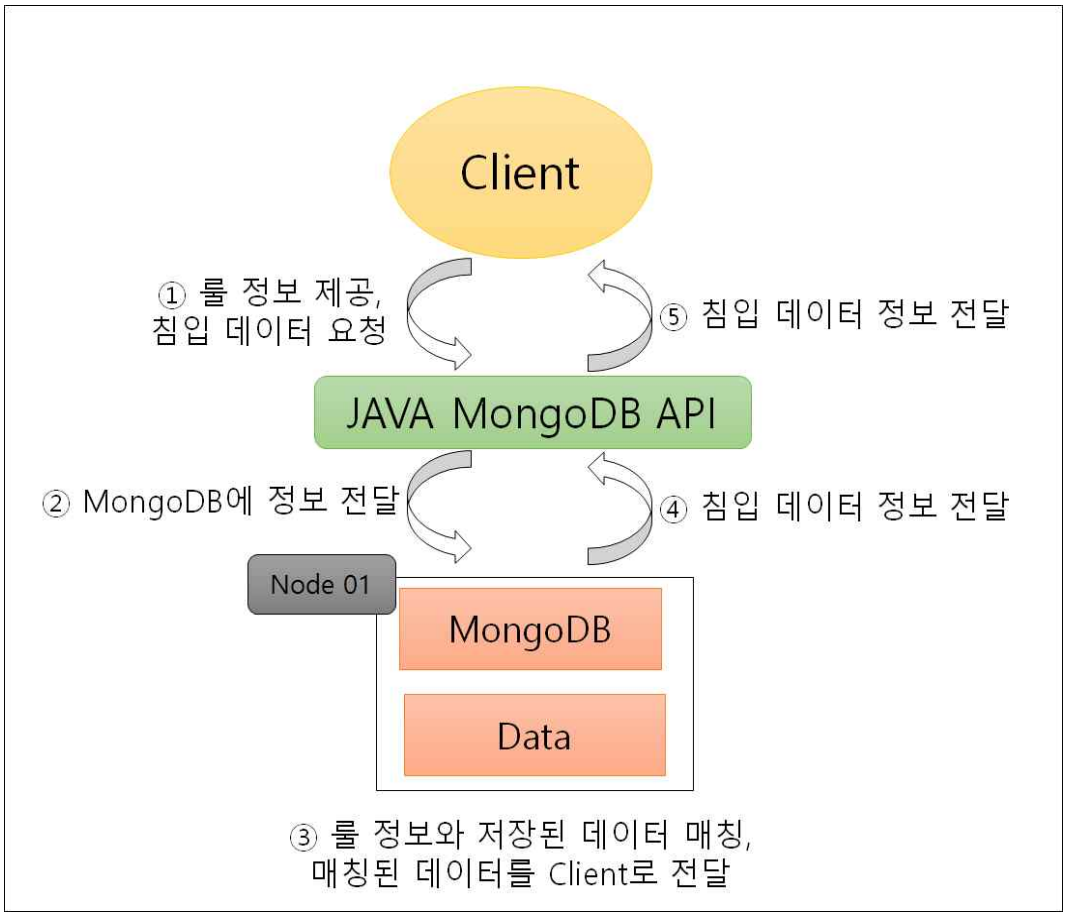
① 관리자는 클라이언트를 이용하여 MongoDB 서버로 침입 데이터에 대한 데이터를 요청하며, 이와 함께 침입 데이터를 판단할 수 있는 Role 정보를 같이 전송

② JAVA MongoDB API는 클라이언트로부터 받은 정보를 MongoDB가 이해할 수 있는 언어로 변환하여 MongoDB에 쿼리 전달

③ MongoDB 데이터베이스는 클라이언트가 JAVA MongoDB API를 통해 전달한 Role 정보를 가지고 저장되어 있는 스노트 시그니쳐 데이터 중 매칭 되는 데이터를 추출하여 클라이언트로 데이터 전송

④ MongoDB 데이터베이스에서 추출된 침입 데이터를 JAVA HBase API가 전달받아 클라이언트에서 이해할 수 있는 언어로 변환하여 클라이언트에 데이터 전달

⑤ 클라이언트는 JAVA MongoDB API로부터 MongoDB 데이터베이스가 전달한 침입데이터를 전달 받으며, 관리자에게 전달받은 데이터가 보안상 위협 요소가 있음을 통지

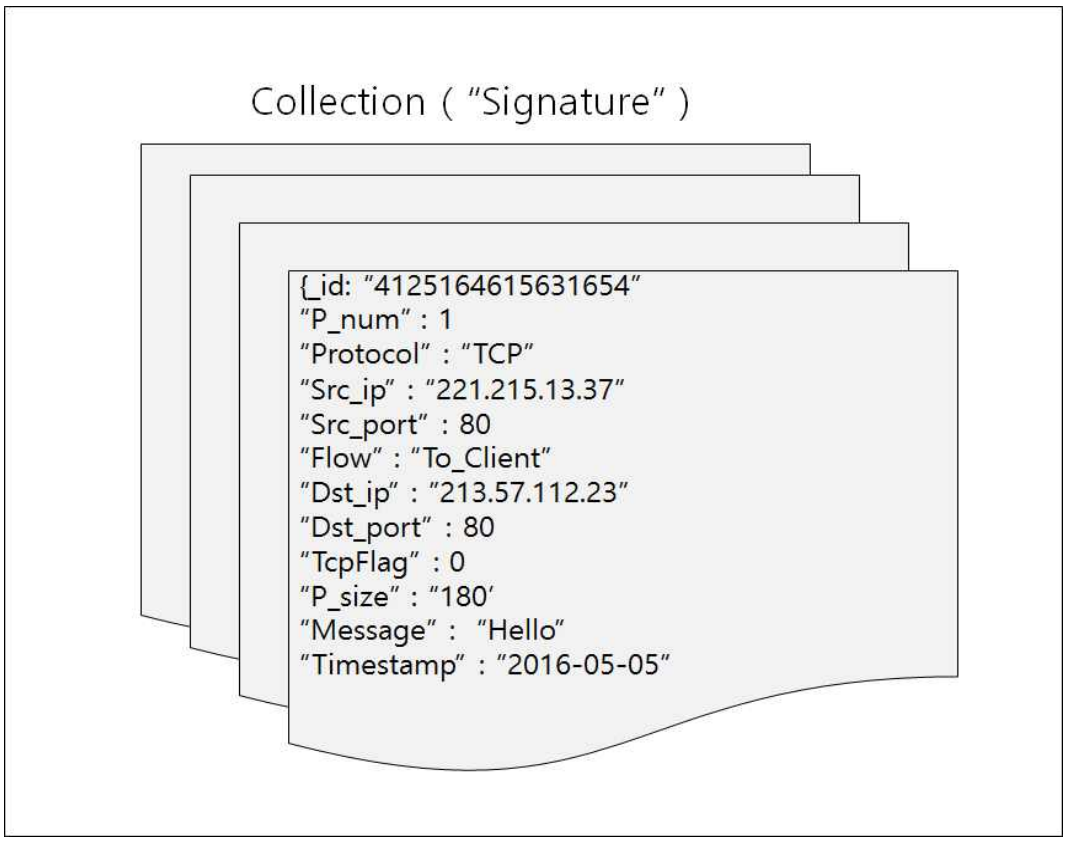


<그림 3-5> Standalone MongoDB Operating Structure

위와 같은 방식으로 단일 노드에서 동작하는 MongoDB 기반의 보안 이벤트 처리 프로토타입 시스템을 구축할 수 있다.

다음으로는 MongoDB 데이터베이스 구축을 하고 데이터베이스에 저장을 할 스노트 시그니쳐 더미 데이터의 생성 방법이다.

MongoDB는 문서 기반 저장 구조이기 때문에, 레코드 간의 관계 설명이 가능하여 개념적으로 RDB와 비슷하게 접근이 가능하다 따라서 RDB에서 MongoDB로 쉽게 이동을 할 수 있다.



<그림 3-6> Snort Signature Data Structure in MongoDB

<그림 3-6>은 MongoDB에서의 스노트 시그니쳐 데이터 구조이다. 하나 씩 살펴보면 다음과 같다.

<표 3-3> MongoDB의 스노트 시그니쳐 데이터 구조

구분	설명
_id	• 저장된 패킷 번호
P_num	• 데이터가 생성된 시간
Protocol	• TCP/IP 프로토콜 정보
Src_ip	• 공격자의 주소
Src_port	• 공격자의 포트
Flow	• 패킷의 이동 방향

Dst_ip	• 목적지의 주소
Dst_port	• 목적지의 포트
TcpFlag	• TCP/IP의 Flag 비트 값
P_size	• 패킷의 크기를
Message	• 패킷에 실린 데이터 내용을 저장
Timestamp	• 패킷이 들어온 시간

위와 같은 구조로 Snort Signataure 기반의 패킷 데이터를 저장하고, 저장된 데이터 중에 침입으로 간주되는 데이터를 추출한다. 클라이언트에서 데이터 추출 명령이 떨어지고, 명령이 데이터베이스에 도달하여 명령이 수행되고 도출된 결과물이 클라이언트까지 도달하는 시간을 측정한다.

제2절 분산처리 기반 처리 방법론

NoSQL 제품의 가장 큰 장점이라고 하면 분산처리 시스템이라고 볼 수 있을 것이다. 제품마다 차이는 있지만 분산 시스템의 장점으로는 고성능, 확장성, 가용성 등 RDBMS에서 제공하기 어려웠던 부분을 NoSQL의 분산 시스템을 통해서 만족시킬 수 있다.

1. Cassandra

1.1. 분산처리 방법론

Cassandra는 데이터 분산과 가용성 면에서 뛰어난 제품이다. Cassandra는 컨시스턴트(consistent) 해싱을 사용하여 데이터를 분산한다. 컨시스턴트 해싱은 메타-데이터 정보를 조회하지 않아도 클러스터에서 키가 저장된 노드를 바로 찾아갈 수 있는 방법이다. 키의 해시 값을 찾고, 이 해시 갓만으로 노드를 찾아갈 수 있다. 컨시스턴트 해싱은 일련의 해시 값을 가상의 링에 순서대로 나열했다고 가정하고 각 노드는 링에서 각자 맡은 범위만 처리하는 방법이다. 만약 노드를 추가하면 특정 노드가 맡고 있던 범위를 분할하여 새

노드에 할당하여 데이터가 균형 있게 분산된다. 노드를 삭제할 때는 링에서 삭제된 노드가 맡고 있던 범위를 인접 노드가 맡고, 서비스 중에 노드를 자유롭게 추가, 삭제 하면서 영향을 받는 노드 수를 최소화 할 수 있다. Cassandra는 마스터가 없이 동작한다. 즉, 데이터 분산이나 복구를 관장하는 별도의 서버가 없기 때문에, SpoF(Single Point of Failure, 단일 고장점)이 없다. 마스터가 없는 대신 각 노드가 주기적으로 서로 메타-데이터 정보를 주고받는다. 이를 가십 프로토콜(gossip protocol)이라 한다. 가십 프로토콜을 통해 노드는 다른 노드가 살아 있는지 알 수 있다.

Cassandra는 정합성 레벨(Consistency Level)을 제공하여 가용성을 높인다. 레벨이 낮으면 노드가 다운되어서 서비스의 다운타임(downtime)이 발생하지 않을 수 있다. 예를 들어, 키에 해당하는 복제 데이터를 저장하는 노드가 3개 있고, 이 중 하나가 다운된다면, 일반적으로는 쓰기 요청이 왔을 때 다운된 노드에 복제 데이터를 쓰지 못하기 때문에 요청 결과에 대해 바로 성공을 반환하지 않는다. 하지만, 레벨을 정족수(quorum)나 1로 설정하면 동작하고 있는 노드의 수가 설정한 수만큼 있는 경우 바로 성공을 반환한다. 따라서 노드3대가 모두 다운되지 않는 한 오류가 발생하지 않는다.

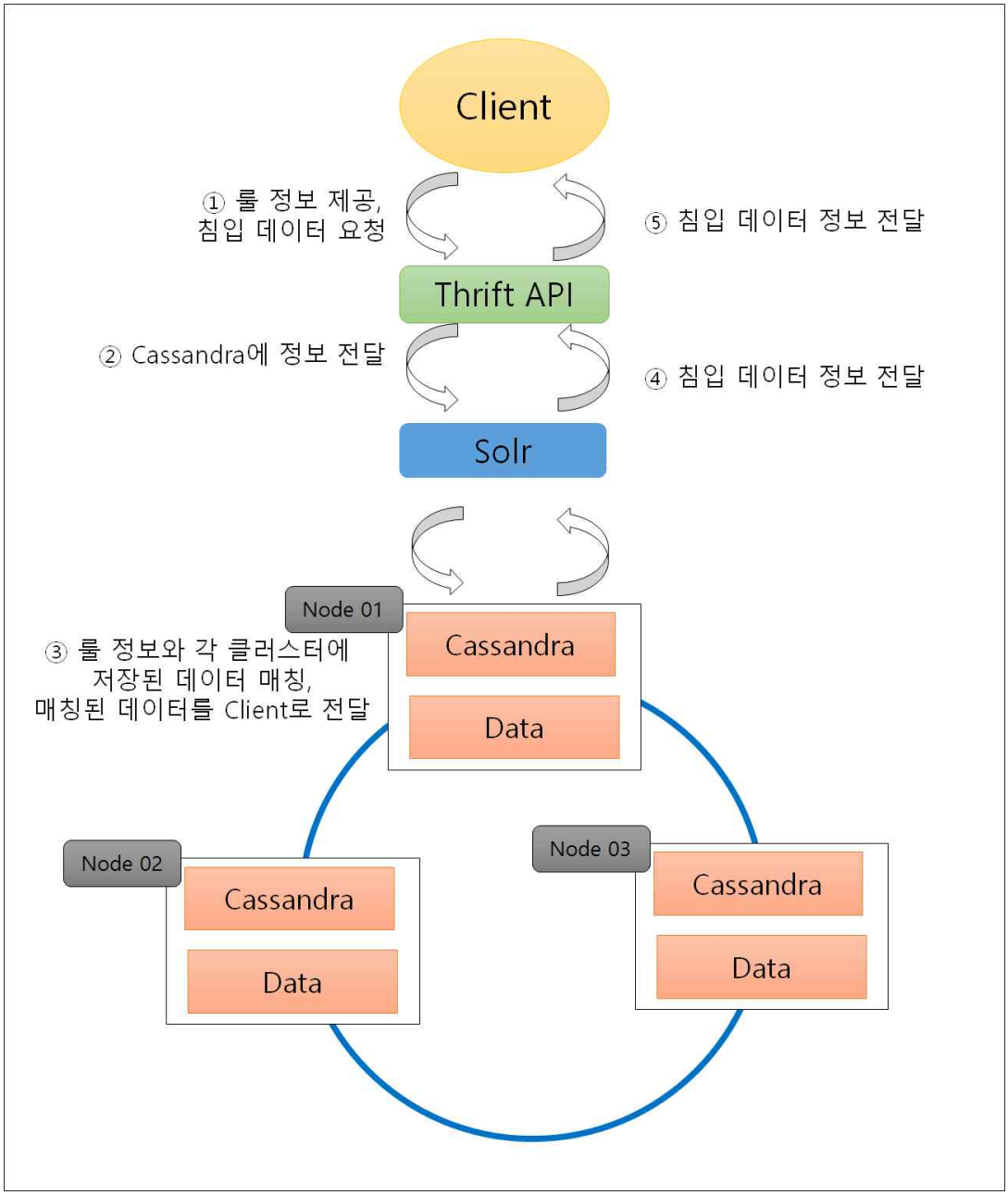
Cassandra는 데이터 복제에 대한 오류에 대해 복구성이 좋고 쉽게 노드를 추가, 삭제 할 수 있는 장점이 있다. 따라서, Cassandra를 이용하여 분산 처리 침입 탐지 시스템 구축 시, 침입 탐지 시스템이 감당해야 할 보안 이벤트의 양의 따라 노드의 개수를 조정하며 최적화된 환경에서 처리할 수 있고, Cassandra의 고가용성으로 시스템 고장 없이 침입을 탐지할 수 있게 된다.

분산처리하는 Cassandra의 성능을 검증하기 위해서 3대의 서버에 Cassandra 데이터베이스를 구축하고, 스노트 시그니쳐 기반의 데이터를 저장한 후 분석하여 분산처리 하는 Cassandra의 성능을 검증하는 방법을 제시한다.

1.2. 빅데이터 처리 프로토타입 시스템 설계

세 대의 노드위에 Cassandra를 설치하고 각각의 데이터베이스에 스노트 시그니쳐 더미 데이터를 생성하여 저장한다. 저장된 스노트 시그니쳐 데이터

를 이용하여 보안이벤트 데이터를 처리하며 그 과정은 <그림 3-7>과 같다.



<그림 3-7> Distributed Processing Operating Structure of Cassandra

① 관리자는 클라이언트를 이용하여 Cassandra 서버에 침입 데이터에 대한 데이터를 요청하며, 이와 함께 침입데이터를 판단할 수 있는 Role 정보를 같이 전송

② Thrift API는 클라이언트로부터 받은 정보를 Cassandra가 이해할 수 있는 언어로 변환하여 Cassandra에 쿼리 전달

③ 검색 플랫폼 Solr가 Thrift API를 통해 전달한 Role 정보를 가지고 각 노드에 저장되어 있는 스노트 시그니쳐 데이터 중 매칭 되는 데이터를 추출하여 클라이언트로 데이터

전송

④ Cassandra 데이터베이스에서 추출된 침입 데이터를 Thrift API가 전달받아 클라이언트에서 이해할 수 있는 언어로 변환하여 클라이언트에 데이터 전달

⑤ 클라이언트는 Thrift API로부터 Cassandra가 전달한 침입데이터를 전달 받으며, 관리자에게 전달 받은 데이터가 보안상 위협 요소가 있음을 통지

위와 같은 방식으로 다중 노드에서 동작하는 Cassandra 기반의 보안 이벤트 처리 프로토타입 시스템을 구축할 수 있다. Cassandra는 분산처리 시 각 노드를 링 형태로 두고 서로 간에 정보를 주고받으며 분산처리 한다.

Snort Signautre를 더미 데이터를 생성하는 방법은 제3장 제1절 1.2에서 설명한 Cassandra 보안 이벤트 데이터 처리 프로토타입 시스템의 더미 데이터 생성법과 동일하다. 클라이언트에서 데이터 추출 명령이 떨어지고, 명령이 Cassandra에 전달되어 분산처리를 통해 추출된 데이터가 클라이언트까지 다시 도달하는 시간을 측정한다.

2. HBase

2.1. 분산처리 방법론

HBase는 하둡의 HDFS와 Map-Reduce를 통해 분산처리된 환경을 구축할 수 있다. HRegionServer는 데이터 분산의 목적으로, HMaster은 HRegionServer의 모니터링 담당을 목적으로, HDFS는 데이터 저장과 복제의 목적으로, Zookeeper는 HMaster의 위치 정보 유지 및 마스터 선출을 목적으로 생성이 된다. 하둡의 가장 큰 장점은 비정형, 반정형의 대규모 데이터를 비용과 시간을 점감하며 처리하고 분석할 수 있다는 것이다. 시스템을 중단하지 않고 장비를 쉽게 추가하거나 삭제할 수 있고, 일부 장비에 장애가 발생하더라도 전체 시스템에 큰 영향을 주지 않는다. 따라서 HBase 분산처리 시스템을 사용하면 고가용성이 보장되는 환경에서 안정적인 침입 탐지 시스템을 구축할 수 있는 것이다. 또한 데이터의 복제 본을 저장하기 때문에 서버에 장애가 발생해도 데이터의 복구가 가능한데, 이 점은 침입으로 판단되는 보안이벤트가 장애로 인해 분실 시 침입을 감지하지 못하여 발생하게 될

손실을 막을 수 있는 장점이 된다. 또한 하둡은 대용량의 배치 처리에 적합하므로, 대용량의 보안 이벤트를 한 번에 처리할 경우에 적합한 환경이라고 볼 수 있다. 데이터를 실시간으로 분석해야 하는 경우에는 부적합한 환경이라고 볼 수 있지만, 보안 이벤트를 한 번에 모아서 분석하는 시스템의 경우에 적합한 환경으로 볼 수 있다.

분산처리 하는 HBase의 성능을 검증하기 위해서 3대의 서버에 Hbase 데이터베이스를 구축하고, 스노트 시그니쳐 기반의 데이터를 HDFS에 저장한 후 분석하여 분산처리 하는 Hbase 의 성능을 검증하는 방법을 제시한다.

2.2. 빅데이터 처리 프로토타입 시스템 설계

세 대의 노드위에 하둡, Zookeeper, HBase를 설치하고 각각의 데이터베이스에 스노트 시그니쳐 더미 데이터를 생성하여 저장한다. 저장된 스노트 시그니쳐 데이터를 이용하여 보안 이벤트 데이터를 처리한다. 처리 과정은 <그림 3-8>과 같다.

① 관리자는 클라이언트를 이용하여 HBase 서버에 침입 데이터에 대한 데이터를 요청하며, 이와 함께 침입 데이터를 판단할 수 있는 Role 정보 동시 전송

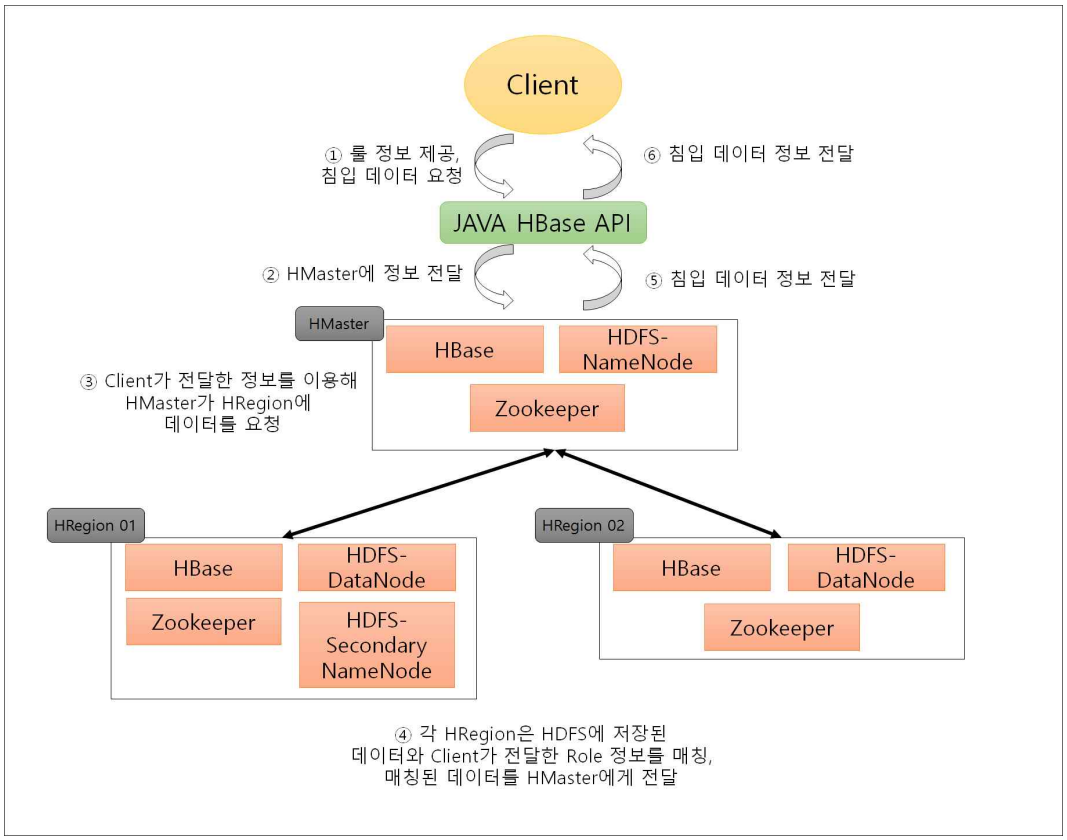
② JAVA HBase API는 클라이언트로부터 받은 정보를 HBase가 이해할 수 있는 언어로 변환하여 HBase에 쿼리 전달

③ HBase는 클라이언트가 JAVA HBase API를 통해 전달한 Role 정보를 가지고 HRegion에 데이터 요청

④ 각 Region은 HDFS에 저장된 스노트 시그니쳐 데이터와 HMaster로부터 받은 Role 정보를 매칭하고, 매칭된 데이터를 다시 HMaster에 전달

⑤ HMaster는 각각의 HRegion으로부터 받은 데이터를 수집하여 클라이언트로 데이터 전송하며, JAVA HBase API는 HMaster로부터 전송된 데이터를 받아 클라이언트에서 이해할 수 있는 언어로 변환하여 클라이언트에 데이터 전달

⑥ 클라이언트는 JAVA HBase API로부터 HMaster가 전달한 침입데이터를 전달 받으며, 관리자에게 전달받은 데이터가 보안상 위협 요소가 있음을 통지



<그림 3-8> Distributed Processing Operating Structure of HBase

위와 같은 방식으로 다중 노드에서 동작하는 HBase 기반의 보안 이벤트 처리 프로토타입 시스템을 구축할 수 있다. HBase는 하둡 프로젝트에 포함되어 있기 때문에 하둡 위에서 HDFS를 통해 분산저장을 할 수 있다.

스노트 시그니쳐 더미 데이터를 생성하는 방법은 제3장 제1절 2.1에서 설명한 HBase 보안 이벤트 데이터 처리 프로토타입 시스템의 더미 데이터 생성법과 동일하다. 클라이언트에서 데이터 추출 명령이 떨어지고, 명령이 HMaster에 도달하여 HRegion에 분산 명령을 내리고 HMaster가 각 HRegion으로부터 데이터를 수집하여 다시 클라이언트에 도달하는 시간을 측정한다.

3. MongoDB

3.1. 분산처리 방법론

MongoDB는 데이터를 분할하여 서로 다른 서버에 나누어 저장하는 샤딩 기능을 이용하여 분산처리 시스템을 구축할 수 있다. MongoDB 는 자동-샤

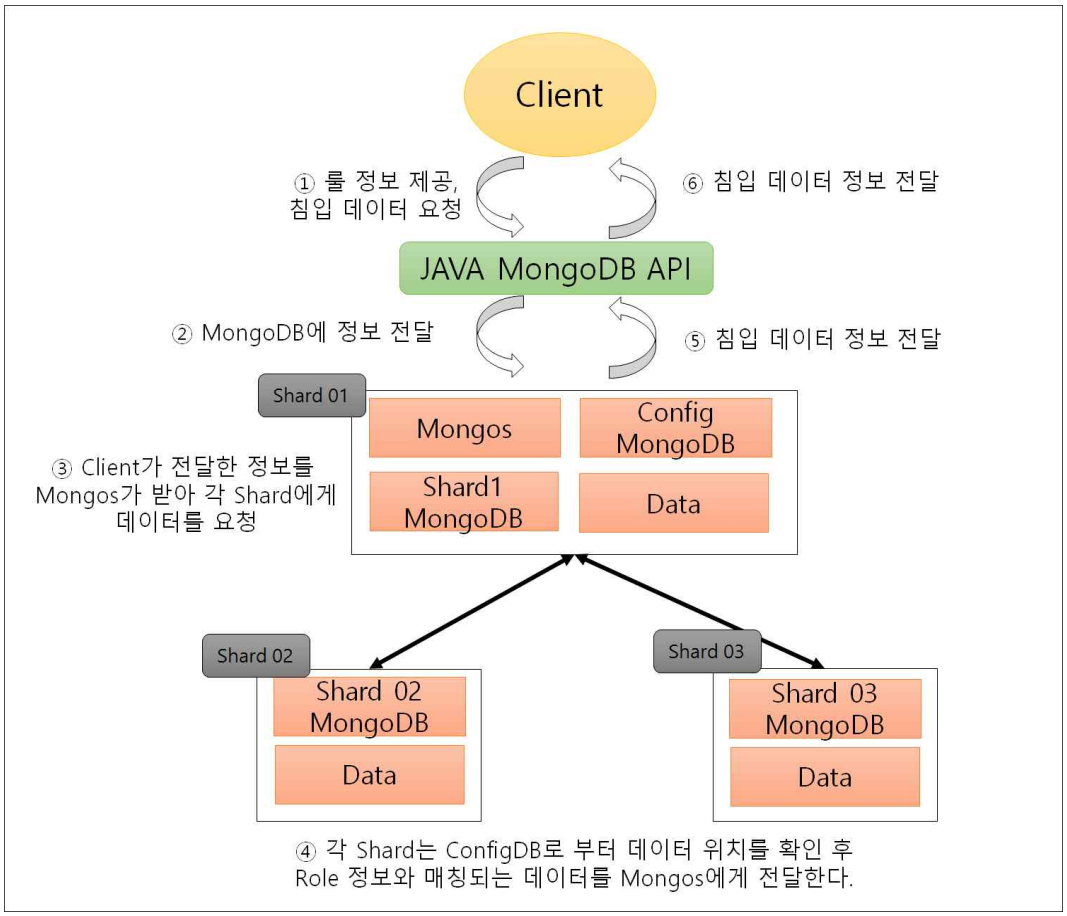
딩을 제공하기 때문에 분산되지 않은 환경에서 작업하다가 용량이 부족하거나 데이터의 분할 저장이 필요할 경우 분산 환경을 바로 구축할 수 있게 된다. 단일 노드에서도 여러 개의 샤드 노드를 만들 수 있고, 여러 개의 서버에 여러 개의 노드를 구축할 수 있다. 시스템의 변화에 적극적으로 변화가 가능한 장점이 있으며, 처리해야할 보안 이벤트의 양에 맞게 자원을 할당 받아서 처리하고 남는 자원은 다른 작업을 처리할 수 있도록 할 수 있다.

MongoDB는 마스터에서 슬레이브로 데이터를 비동기 방식으로 복제를 한다. 비동기 복제의 장점은 복제로 인한 마스터의 성능 저하가 거의 없고, 슬레이브를 추가해도 서비스 성능이 떨어지지 않는 다는 장점이 있다. 하지만 데이터가 대부분 불일치하기 때문에 장애가 생기면 데이터 유실이 발생할 수 있다. MongoDB는 장애가 발생할 시 마스터를 선출하는 방식을 사용한다. 예를 들어, 노드 3대를 각각 Master, Slave1, Slave2로 구성하고 마스터 프로세스를 중단하면, MongoDB는 자동으로 슬레이브 중 하나를 마스터로 선출한다. 이 때 장애가 발생 한 후 마스터를 선출하기 까지 수 초 정도 걸리는데, 다운타임이 아주 짧다는 것을 말한다. 따라서, 신속하게 보안이벤트를 분석해야하는 경우 장애나 시설 증설로 인한 성능 저하가 거의 없는 MongoDB를 사용할 시 유용할 것으로 보인다.

분산처리하는 MongoDB의 성능을 검증하기 위해서 3대의 서버에 MongoDB 데이터베이스를 구축하고, 스노트 시그니쳐 기반의 데이터를 저장한 후 분석하여 분산처리 하는 MongoDB의 성능을 검증하는 방법을 제시한다.

3.2. 빅데이터 처리 프로토타입 시스템 설계

세 대의 노드위에 MongoDB를 설치하고 자동-샤딩을 통해 각각의 데이터베이스에 스노트 시그니쳐 더미 데이터를 생성하여 저장한다. 저장된 스노트 시그니쳐 데이터를 이용하여 보안 이벤트 데이터를 처리한다. 처리 과정은 <그림 3-9>와 같다.



<그림 3-9> Distributed Processing Operating Structure of MongoDB

① 관리자는 클라이언트를 이용하여 MongoDB 서버에 침입 데이터에 대한 데이터를 요청하며, 이와 함께 침입 데이터를 판단할 수 있는 Role 정보 동시 전송

② JAVA MongoDB API는 클라이언트로부터 받은 정보를 MongoDB가 이해할 수 있는 언어로 변환하여 MongoDB에 쿼리 전달

③ MongOS는 클라이언트가 JAVA MongoDB API를 통해 전달한 Role 정보를 가지고 각각의 샤드 MongoDB에 데이터 요청

④ 각 샤드 MongoDB는 Config Server에 저장된 Meta Data와 각 노드에 저장된 스노트 시그니쳐 데이터를 Mongos로부터 받은 Role 정보와 매칭하고, 매칭된 데이터를 다시 MongOS에 전달

⑤ MongOS는 각각의 샤드 MongoDB로부터 받은 데이터를 수집하여 클라이언트로 데이터 전송하며, JAVA MongoDB API는 MongOS로부터 전송된 데이터를 받아 클라이언트에서 이해할 수 있는 언어로 변환하여 클라이언트에 데이터 전달

⑥ 클라이언트는 JAVA MongoDB API로부터 Mongos가 전달한 침입데이터를 전달 받으

며, 관리자에게 전달받은 데이터가 보안상 위협 요소가 있음을 통지

위와 같은 방식으로 다중 노드에서 동작하는 MongoDB 기반의 보안 이벤트 처리 프로토타입 시스템을 구축할 수 있다. MongoDB는 자동-샤당을 통해서 Mongos가 각 샤드 MongoDB에 데이터를 분산 하여 저장할 수 있고, 저장된 데이터를 분산처리 할 수 있다.

스노트 시그니쳐 더미 데이터를 생성하는 방법은 제4장 제1절 3.1에서 설명한 MongoDB 보안 이벤트 데이터 처리 프로토타입 시스템의 더미 데이터 생성법과 동일하다. 클라이언트에서 데이터 추출 명령이 떨어지고, 명령이 Mongos에 도달하여 샤드 MongoDB에 분산 명령을 내리고 Mongos가 각 샤드 MongoDB부터 데이터를 수집하여 다시 클라이언트에 도달하는 시간을 측정한다.

제4장 성능 테스트 및 결과

제1절 테스트 환경 구성

실제 테스트에 앞서 테스트에 필요한 환경 구성에 대해 설명한다. 비약적으로 증가하고 네트워크 트래픽 중 사이버 침해시도라 판단되는 보안이벤트에 대한 수집 역할을 수행하는 가상 센서를 리눅스 기반의 가상화 환경으로 구축하고, 스노트 시그니쳐 기반의 더미 데이터 트래픽의 수량을 늘려가며 테스트 환경에 흘려보낸다. <표 4-1>은 단일 노드와 다중 노드 환경에서의 시스템 구축을 위한 테스트 시스템의 사양이다.

<표 4-1> Test System Specification

항목	사양
CPU	I3-4170 (4 Core)
Memory	16GB
HDD	2TB
OS	Windows 10 Pro x64

<표 4-2>는 단일 노드 보안 이벤트 데이터 처리 프로토타입 시스템을 구축한 가상화 환경의 사양이다.

<표 4-2> Single Node System Specification

항목	사양
CPU	I3-4170 (3 Core)
Memory	12GB
HDD	600GB
OS	CentOS 7 x64

위 표와 같은 사양으로 각 NoSQL 기반의 보안 이벤트 데이터 처리 프로토타입 시스템과 MySQL 기반의 보안 이벤트 데이터 처리 프로토타입 시스템 환경을 구성하였다.

<표 4-3>은 다중 노드 보안 이벤트 데이터 처리 프로토타입 시스템을 구축한 가상화 환경의 사양이다.

<표 4-3> Multi Node System Specification

항목	사양
CPU	I3-4170 (1 Core)
Memory	4GB
HDD	200GB
OS	CentOS 7 x64

다중 노드 환경은 총 3대의 노드로 구성이 되어있으며, 각 노드 당 위의 표와 같은 사양을 가지고 있다. 따라서 단일 노드 환경과 다중노드 환경에서의 사양을 유사하게 맞췄다는 것을 알 수 잇다.

<표 4-4>는 실험에서 사용한 각 데이터베이스와 클라이언트, API의 구성이다.

<표 4-4> Configuration of Database, Client and API

항목	이름	버전
MySQL 데이터베이스	MySQL	5.6.32
NoSQL 데이터베이스	Cassandra	3.0.8
	HBase	1.2.3
	MongoDB	2.6.12
하둡	Hadoop	2.7.1
클라이언트	Java	1.8.0_101
Zookeeper	Zookeeper	3.4.8
검색 엔진	Solr	6.2.1
데이터베이스 API	Thrift API	3.0.8
	HBase API	1.2.3
	MongoDB API	3.3.0
	MySQL API	5.1.39

제2절 테스트 결과

1. 단일 노드 기반의 보안 이벤트 데이터 처리 프로토타입 시스템

먼저 단일 노드 기반의 보안 이벤트 데이터 처리 프로토타입 시스템의 성능을 평가한다.

1.1. Cassandra 기반 성능 테스트 결과

다음 표는 Cassandra 데이터베이스에 각 천만 건, 이천만 건, 일억 건의 스노트 시그니처 더미 데이터를 저장하고 그 중 침입으로 판단되는 데이터를 추출하는데 걸린 수행 시간이다. 침입 데이터는 각 총 데이터 수에 10분의 1을 차지한다.

<표 4-5> Single Node Cassandra Performance Table

총 데이터 수(packet)	추출 데이터 수(packet)	수행시간(sec)
10,000,000	1,000,000	41.112
20,000,000	2,000,000	87.704
100,000,000	10,000,000	487.771

1.2. HBase 기반 성능 테스트 결과

다음 표는 HBase 데이터베이스에 각 천만 건, 이천만 건, 일억 건의 스노트 시그니처 더미 데이터를 저장하고 그 중 침입으로 판단되는 데이터를 추출하는데 걸린 수행 시간이다. 침입 데이터는 각 총 데이터 수에 10분의 1을 차지한다.

<표 4-6> Single Node HBase Performance Table

총 데이터 수(packet)	추출 데이터 수(packet)	수행시간(sec)
10,000,000	1,000,000	51.981
20,000,000	2,000,000	118.678
100,000,000	10,000,000	535.224

1.3. MongoDB 기반 성능 테스트 결과

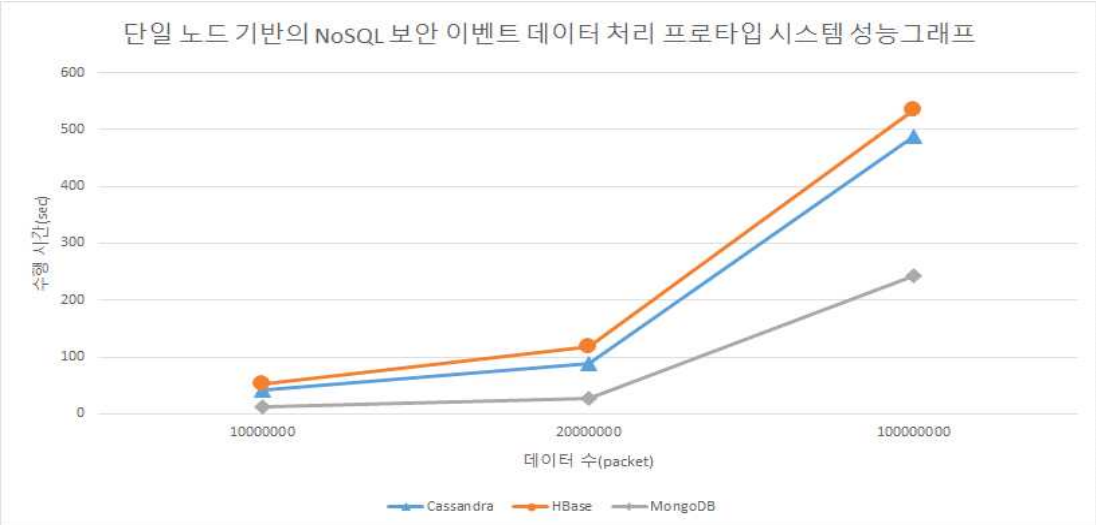
다음 표는 MongoDB 데이터베이스에 각 천만 건, 이천만 건, 일억 건의 스노트 시그니처 더미 데이터를 저장하고 그 중 침입으로 판단되는 데이터를 추출하는데 걸린 수행 시간이다. 침입 데이터는 각 총 데이터 수에 10분의 1을 차지한다.

<표 4-7> Single Node MongoDB Performance Table

총 데이터 수(packet)	추출 데이터 수(packet)	수행시간(sec)
10,000,000	1,000,000	13.244
20,000,000	2,000,000	26.249
100,000,000	10,000,000	243.308

1.4. 단일 노드 기반 NoSQL 성능 비교

<그림 4-1>은 MySQL을 제외한 3가지 NoSQL 데이터베이스 기반의 보안 이벤트 데이터 처리 프로토타입 시스템의 성능을 비교한 그래프이다.



<그림 4-1> Single Node based NoSQL Prototype System Performance Comparison Graph

3개의 단일 노드에서 구축한 NoSQL 프로토타입 시스템 중 가장 우수한 성능을 보이는 데이터베이스는 MongoDB로 결과가 나왔다. MongoDB를 이어 Cassandra와 HBase의 순으로 성능을 나타내고 있으며, MongoDB와는 다소 성능의 차이가 나는 것을 볼 수 있다.

2. 다중 노드 기반의 보안 이벤트 데이터 처리 프로토타입 시스템

다음으로 다중 노드 기반의 보안 이벤트 데이터 처리 프로토타입 시스템의 성능을 평가한다.

2.1. Cassandra 기반 성능 테스트 결과

<표 4-8>는 Cassandra 데이터베이스에 각 천만 건, 이천만 건, 일억 건의 스노트 시그니처 더미 데이터를 저장하고 그 중 침입으로 판단되는 데이터를 추출하는데 걸린 수행 시간이다. 침입 데이터는 각 총 데이터 수에 10분의 1을 차지한다.

<표 4-8> Multi Node Cassandra Performance Table

총 데이터 수(packet)	추출 데이터 수(packet)	수행시간(sec)
10,000,000	1,000,000	29.576
20,000,000	2,000,000	58.247
100,000,000	10,000,000	394.619

2.2. HBase 기반 성능 테스트 결과

<표 4-9>는 HBase 데이터베이스에 각 천만 건, 이천만 건, 일억 건의 스노트 시그니처 더미 데이터를 저장하고 그 중 침입으로 판단되는 데이터를 추출하는데 걸린 수행 시간이다. 침입 데이터는 각 총 데이터 수에 10분의 1을 차지한다.

<표 4-9> Multi Node HBase Performance Table

총 데이터 수(packet)	추출 데이터 수(packet)	수행시간(sec)
10,000,000	1,000,000	21.287
20,000,000	2,000,000	34.148
100,000,000	10,000,000	236.254

2.3. MongoDB 기반 성능 테스트 결과

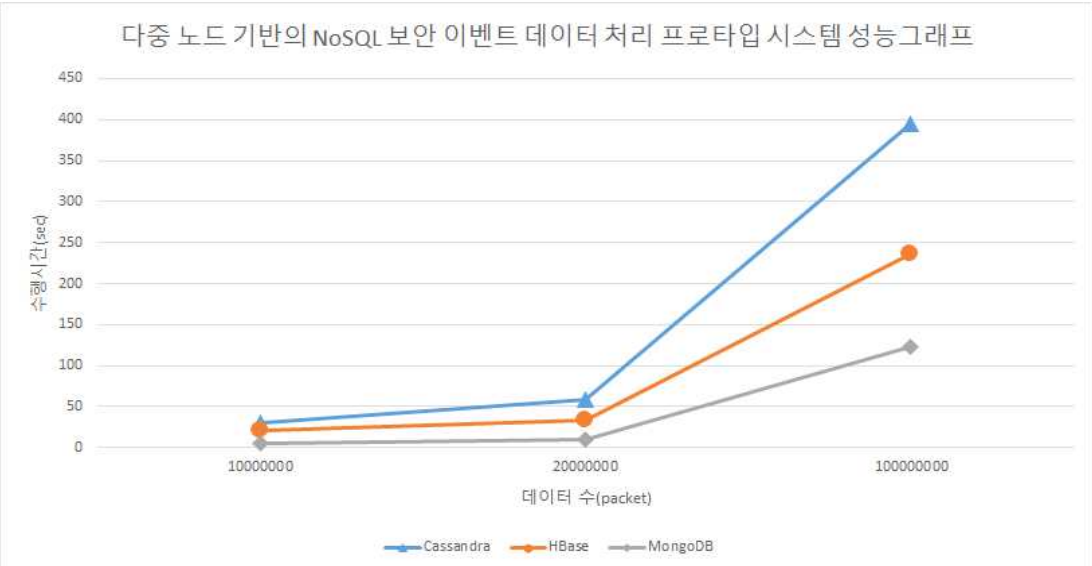
<표 4-10>은 MongoDB 데이터베이스에 각 천만 건, 이천만 건, 일억 건의 스노트 시그니처 더미 데이터를 저장하고 그 중 침입으로 판단되는 데이터를 추출하는데 걸린 수행 시간이다. 침입 데이터는 각 총 데이터 수에 10분의 1을 차지한다.

<표 4-10> Multi Node MongoDB Performance Table

총 데이터 수(packet)	추출 데이터 수(packet)	수행시간(sec)
10,000,000	1,000,000	5.623
20,000,000	2,000,000	9.774
100,000,000	10,000,000	122.377

2.4. 다중 노드 기반 NoSQL 성능 비교

다음 <그림 4-2>는 NoSQL 데이터베이스 기반의 보안 이벤트 데이터 분산 처리 프로토타입 시스템의 성능을 비교한 그래프이다.



<그림 4-2> Multi Node based NoSQL Prototype System Performance Comparison Graph

단일 노드에서의 결과와 마찬가지로 MongoDB 기반의 프로토타입 시스템의 성능이 다른 2개의 데이터베이스보다 우수한 성능을 보이는 것을 볼 수

있다. MongoDB를 이어 HBase, Cassandra 순이다. 단일 노드와는 다르게 Cassandra와 HBase 모두 다중 노드에서 분산 처리할 때 상당히 성능이 증가한 것을 볼 수 있지만, HBase의 성능 증가폭이 더 커서 Cassandra의 성능을 역전하였다.

3. MySQL 기반의 보안 이벤트 데이터 처리 프로토타입 시스템

위 의 실험 결과를 통해 단일 노드에서 동작하는 보안 이벤트 데이터 처리 프로토타입 시스템과, 다중 노드에서 동작하는 보안 이벤트 데이터 처리 프로토타입 시스템에서 가장 우수하게 동작하는 시스템은 MongoDB 기반의 보안 이벤트 데이터 처리 프로토타입 시스템임을 알 수 있었다.

기존 침입 탐지 시스템의 유형인 MySQL 기반의 보안이벤트 데이터 처리 프로토타입 시스템 성능과 MongoDB 기반의 보안이벤트 데이터 처리 프로토타입 시스템 성능의 비교를 통해서 가장 우수한 성능을 보이는 시스템을 제안한다.

3.1. MySQL 기반 성능 테스트 결과

<표 4-11>은 MySQL 데이터베이스에 각 천만 건, 이천만 건, 일억 건의 스노트 시그니처 더미 데이터를 저장하고 그 중 침입으로 판단되는 데이터를 추출하는데 걸린 수행 시간이다. 침입 데이터는 각 총 데이터 수에 10분의 1을 차지한다.

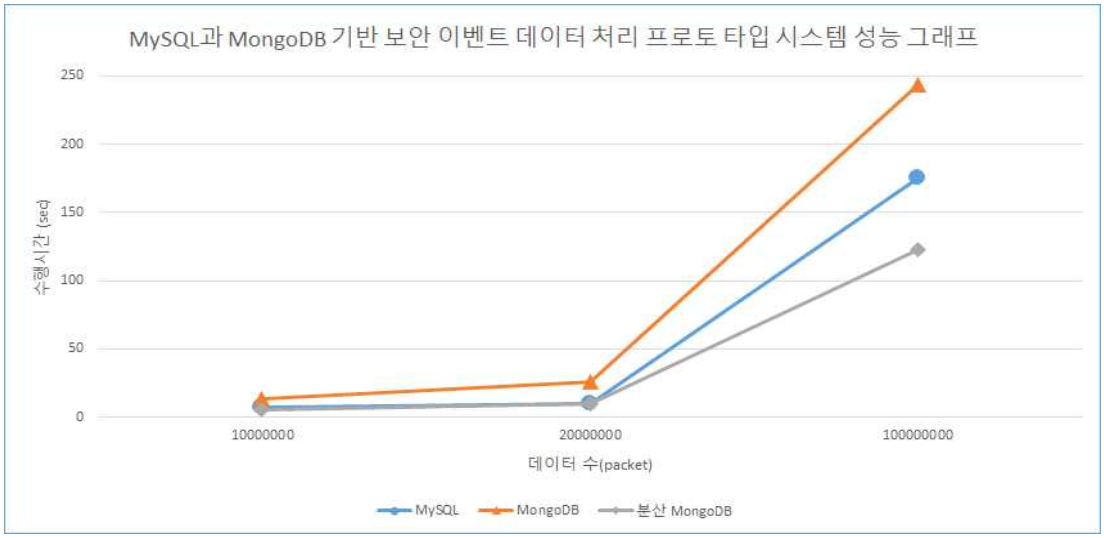
<표 4-11> Single Node MySQL Performance Table

총 데이터 수(packet)	추출 데이터 수(packet)	수행시간(sec)
10,000,000	1,000,000	6.770
20,000,000	2,000,000	9.787
100,000,000	10,000,000	175.538

3.2. MySQL 및 NoSQL 기반 테스트 성능 비교

다음 <그림 4-3>은 MySQL 데이터베이스 기반의 보안 이벤트 데이터 처리 프로토타입 시스템과 NoSQL 기반의 보안 이벤트 데이터 처리 프로토타입 시스템 중 가장 우수한 성능을 보인 MongoDB기반의 보안 이벤트 데이터 처리 프로토타

입 시스템과, MongoDB기반의 보안 이벤트 데이터 분산처리 프로토타입 시스템의 성능을 비교한 그래프이다.



<그림 4-3> MySQL and MongoDB based Prototype System Performance Comparison Graph

그래프를 분석해보면, 다중 노드 환경에서 분산처리 하는 MongoDB기반의 보안 이벤트 데이터 처리 프로토타입 시스템이 가장 우수하고, 뒤를 이어 MySQL, 단일 노드에서 동작하는 MongoDB 순으로 나타난다. 단일 노드에서 동작하는 MongoDB 기반의 보안 이벤트 데이터 처리 프로토타입 시스템의 경우에는 다른 두 시스템보다는 조금 떨어지는 성능을 보이고, 분산처리 하는 MongoDB와 MySQL의 경우에는 천만 건, 이천만 건에서는 크게 차이를 보이지 않고 분산처리 하는 MongoDB가 조금 우세하게 나타나지만, 데이터의 양이 증가할수록 성능에 격차가 생기고 있다. 일억 건에서는 MySQL이 '175.538초', 분산처리 하는 MongoDB는 '122.377초' 로 많은 차이가 나는 것을 확일 할 수 있었다.

제5장 결론

일반적인 데이터 추출의 경우에는 'Full Text Search'를 진행하기 때문에 NoSQL이 인덱싱(Indexing) 처리를 통해 데이터 추출 속도를 월등히 증가 시킬 수 있다. 하지만, 침입 탐지 특성상 패킷에 저장되어있는 내용 중 일부분이 매치되었을 경우 침입으로 간주하기 때문에 'Full Text Search'가 아닌 'Partial Word Search'를 진행하게 된다.

따라서, MySQL과 NoSQL 데이터베이스 모두 데이터를 추출하기 위해 모든 데이터들을 확인해야하고, NoSQL은 인덱싱의 효과를 누릴 수 없기 때문에, NoSQL의 성능이 월등히 증가 하지 못하게 된다.

위와 같은 이유로 단일 노드 기반의 프로토타입 시스템 중에서는 MySQL이 가장 우수한 성능을 보이는 것으로 나타난다. 하지만 같은 사양의 단일 노드를 다중 노드로 분산하여 처리할 경우 NoSQL 기반의 프로토타입 시스템의 성능이 증가하여, 최종적으로 MongoDB 기반의 분산처리 환경의 프로토타입 시스템이 가장 우수한 것으로 나타났다.

적은 양의 데이터일 때는 MySQL과 MongoDB 분산 처리 환경의 차이가 미미하였지만, 데이터의 양이 커질수록 성능의 차이가 분명하게 나타나는 것을 확인할 수 있었는데, 데이터의 양이 이억 건, 삼억 건 등으로 증가할수록 그 차이는 더욱 커져 MySQL의 성능이 굉장히 저하될 것으로 판단된다. 이는 전체 침해위협 탐지시스템 병목 현상의 주요 원인으로 발생될 수 있으며, 이를 해결하기 위해 NoSQL 기술을 적용함으로써 빅데이터화 되고 있는 침해위협들을 보다 신속하게 처리 할 수 있을 것이다.

참고 자료

[1] Secure coding, http://cwe.mitre.org

[2] JAVA, http://wikisecurity.net/guide:java

[3] MS, https://technet.microsoft.com/library/security/ms13-078

[4] MS, http://insecure.org/sploits/Microsoft.frontpage.insecurities.html

[5] REDHAT, https://securityblog.redhat.com/2014/10/15/poodle-a-ssl3-vulnerability-cve-2014-3566/

[5] APACHE, http://www.apache.org/