다변량 데이터 스트림을 위한 아파치 스톰 기반 질의 필터링 시스템

김영국°, 손시운, 문양세
강원대학교 컴퓨터과학과
e-mail: {ygkim, ssw5176, ysmoon}@kangwon.ac.kr

Apache Storm based Query Filtering System for Multivariate Data Streams

Youngkuk Kim°, Siwoon Son, and Yang-Sae Moon
Dept. of Computer Science, Kangwon National University

요 약

최근 빠르게 발생하는 빅데이터 스트림이 다양한 분야에서 활용되고 있다. 이러한 빅데이터 전체를 수집하고 처리하는 것은 매우 비경제적이므로, 데이터 스트림 중 필요한 데이터를 걸러내는 필터링 과정이 필요하다. 본 논문에서는 아파치 스톰(Apache Storm)을 사용하여 데이터 스트림의 질의 필터링 시스템을 구축한다. 스톰은 대용량 데이터 스트림을 처리하기 위한 실시간 분산 병렬 처리 프레임워크이다. 하지만, 스톰은 입력 데이터 구조나 알고리즘 변경 시, 코드의 수정과 재배포, 재시작 등이 필요하다. 따라서, 본 논문에서는 이 같은 문제를 해결하기 위해 아파치 카프카(Apache Kafka)를 사용하여 데이터 수집 모듈과 스톰의 처리 모듈을 분리함으로써 시스템의 가용성을 크게 높인다. 또한, 시스템을 웹 기반 클라이언트-서버 모델로 구현하여 사용자가 언제 어디에서든 질의 필터링 시스템을 사용할 수 있게 하며, 웹 클라이언트를 통해 입력한 질의를 자동적 분석하는 쿼리 파서를 구현하여 별도의 프로그램의 수정 없이 질의 필터링을 적용할 수 있다.

1. 서론

최근 SNS, 금융, 사물인터넷 등에서 빅데이터 스트림이 빠르게 생성되고 있다. 데이터 스트림이란 지속적으로 끊임없이 생성되는 데이터이며, 이러한 데이터 스트림 전체를 저장하고 처리하는 것은 매우 비효율적이다. 따라서, 데이터 스트림에서 불필요한 데이터를 제거하거나 데이터의 패턴을 파악하기 위해 데이터 스트림을 지속적으로 모니터링하고 필터링하는 것이 효과적이다. 하지만, 데이터 스트림의 생성 속도와 크기가 점점 증가하면서 단일 서버에서 모든 데이터를 실시간 처리하는 것은 한계가 있다.

빅데이터[1]의 특징 중 하나인 속도(velocity)는 기존 시스템으로 처리하기 힘들 정도로 데이터가 빠르게 생성됨을 의미한다. 이를 위해, 데이터 스트림을 분산 환경에서 처리하기 위한 연구가 활발하다. 대표적인 데이터 스트림 분산 처리 프레임워크로는 아파치 스톰(Apache Storm)[2]과 아파치 스파크 스트리밍(Apache Spark Streaming)[4] 등이 있다. 특히, 스톰은 대용량 데이터 스트림을 분산된 서버에서 실시간으로 처리할 수 있는 분산 병렬 프레임워크이다. 이러한 분산 처리 시스템은 데이터 스트림을 처리하는데 초점을 두며, 데이터 스트림의 입출력을 위해 별도의 큐잉 시스템과 함께 사용한다. 아파치 카프카(Apache Kafka)[5]는 분산 메시지 큐잉 시스템으로 실시간 로그 처리에 특화되어 있어, 데이터 스트림 처리 프레임워크의 입력이나 파이프라인으로 활용된다[7].

본 논문에서는 다변량 데이터 스트림 처리를 위한 스톰 기반 질의 필터링 시스템을 설계하고 구현한다. 제안하는 시스템은 클라이언트-서버 모델로 구성한다. 클라이언트는 사용자에게 웹 UI를 제공하며, 사용자는 클라이언트를 통해 데이터 스트림의 수집, 질의, 출력의 작업을 시각적으로 정의 및 수행할 수 있다. 먼저, 사용자는 웹 UI에서 데이터 스트림이 생성되는 소스(source) 정보와 구조를 정의한다. 다음으로, 처리된 데이터 스트림을 출력할 저장소인 데스티네이션(destination) 정보를 입력한다. 그런 다음, 입력 데이터 스트림에 적용할 질의를 생성한다. 마지막으로, 질의 필터링과 데이터 수집 작업을 수행할 수 있다.

서버는 데이터베이스, 카프카, 스톰 클러스터로 구성된다. 서버는 웹 UI에서 입력된 명령어와 정보를 데이터베이스에 저장하고, 사용자의 데이터 스트림을 수집하여 카프카에 저장한다. 사용자가 질의 필터링 실행 명령을 보내면 스톰 클러스터에 명령을 전달하고, 스톰 클러스터는 카프카로 입력되는 데이터 스트림에 질의 필터링을 적용한다.

기존 스톰은 데이터 스트림의 수집과 처리를 함께 정의하기 때문에, 데이터 구조 또는 처리 알고리즘의 변경이 필요할 경우 프로그램의 소스코드를 수정하고 재배포해야하는 문제가 있다. 이 문제로 인해 끊임없이 생성되는 데이터 스트림은 프로세스가 정상적으로

* 이 논문은 2018 년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No. 2016-0-00179, 데이터 스트림 정제를 위한 지능형 샘플링 및 필터링 기술 개발).

재구동될 때까지 모두 유실될 수 있다. 이를 해결하기 위해, 제안 시스템에서는 데이터 수집 모듈과 스톰의 처리 모듈을 분리하여 가용성을 높인다. 데이터 수집 모듈은 사용자가 입력한 소스로부터 데이터 스트림을 읽어 시스템의 카프카로 입력하며, 스톰은 카프카에 저장된 데이터 스트림에 필터링을 적용한다. 처리된 데이터는 사용자가 지정한 저장소로 출력된다.

2. 관련 연구

필터링[8]은 데이터 신뢰도 및 분석 속도 향상을 위해, 데이터에서 주어진 조건을 만족하는 데이터만을 추출하는 기법이다. 대표적으로, 사용자의 질의를 만족하는 데이터만 추출하는 질의(Query) 필터링, 데이터가 집합에 속하는지 여부를 확률적으로 계산하는 블룸(Bloom) 필터링, 센서 데이터처럼 오차가 포함되어 있는 데이터로부터 원본 데이터를 추정하는 칼만(Kalman) 필터링[9] 등이 있다. 이러한 기법들을 데이터 스트림에 적용하는 방법은 크게 두 가지로 분류된다. 먼저, 고정 쿼리(standing queries) 방식은 데이터가 입력되었을 때, 미리 정의된 질의 조건을 만족하는지 판단하고, 데이터가 조건을 만족하면 저장하거나 통과시킨다. 다음으로, 애드 혹(ad-hoc) 방식은 입력되는 데이터를 일정 시간이나 크기만큼(혹은 전체를) 저장하고 질의를 처리하는 방식이다. 본 논문에서는 이중 간단한 고정 쿼리 방식의 질의 필터링을 구현한다. 질의 필터링은 데이터 스트림 환경에서 적용 가능하며, 데이터 스트림을 분류하거나 원하는 데이터만을 추출할 수 있어 널리 사용되는 방법이다.

아파치 스톰[2]은 데이터 스트림을 실시간 처리하기 위한 대표적인 분산 실시간 처리 시스템이다. 스톰은 데이터의 입력부터 출력까지 일련의 작업을 토폴로지(topology)로 표현하고, 토폴로지는 다수의 스파우트(spout)와 볼트(bolt)로 구성된다. 그림 1은 이러한 스톰의 동작 과정을 보여준다. 먼저, 스파우트는 데이터 스트림의 발생지로부터 데이터를 입력 받아 스톰에서 사용하는 데이터 형태인 튜플(tuple)로 바꾸어 볼트에게 전달한다. 다음으로, 볼트는 스파우트 또는 볼트로부터 전달받은 튜플을 처리하여 다음 볼트로 전달하거나, 결과를 출력 또는 저장 장치로 보낸다. 스톰은 입력 데이터의 구조가 바뀔 경우 소스 코드를 수정해야 하는 문제가 생긴다. 즉, 소스 코드가 수정되면 새로 빌드하고 토폴로지를 재배포해야 하므로, 서비스가 중단되는 문제가 있다.

그림 1. 아파치 스톰의 데이터 스트림 처리 과정.

아파치 카프카[5]는 대용량으로 발생하는 실시간 로그 데이터 처리에 특화된 분산 메시지 큐잉 시스템이다. 카프카는 분산 구조로 설계되어 시스템 확장이 매우 쉽고 처리 속도가 빠르다. 또한, 로그 데이터를 메모리가 아닌 디스크에 저장함으로써 인메모리 기반 메시지 큐에 비해 높은 데이터 안정성을 보장하며, 메시지 큐 역할과 로그 수집기 역할을 함께 수행 가능하다는 특징이 있다. 카프카는 발행-구독 모델을 기반으로 동작하며, 프로듀서, 컨슈머, 브로커로 구성된다. 프로듀서(producer)는 다른 시스템에서 발생한 메시지를 브로커로 발행(publish)하는 역할을 하며, 컨슈머(consumer)는 다른 시스템에서 브로커로부터 메시지를 구독(subscribe)하는 역할을 한다. 이 때, 브로커는 메시지들을 토픽(topic)별로 분류하여 적재한다.

3. 아파치 스톰 기반 질의 필터링 시스템 설계

본 절에서는 제안하는 아파치 스톰 기반 질의 필터링 시스템을 설계한다. 제안하는 시스템은 그림 2와 같이 클라이언트-서버 모델로 구성된다. 클라이언트-서버 구조는 대부분의 분석 및 관리 시스템에서 사용하는 네트워크 기반 모델로서, 유지 보수가 용이하고 다수의 사용자가 동시에 사용 가능한 장점이 있다. 특히, 본 시스템은 클라이언트를 웹 기반으로 구현하여, 사용자가 웹 브라우저를 통해 언제 어디서든 질의 필터링 시스템을 쉽게 활용할 수 있다. 또한, 기존 스톰의 문제점을 해결하기 위해 데이터 스트림의 입력을 관리하는 소스와 출력을 관리하는 데스티네이션을 구현한다. 소스는 데이터 발생지의 정보를 관리하고, 데이터 스트림을 수집하는 역할을 한다. 반대로 데스티네이션은 처리된 데이터 스트림을 출력하는 저장소의 정보를 관리하고, 데이터 스트림을 내보내는 역할을 한다. 스톰 토폴로지는 입력된 질의에 따라 질의 필터링만 수행하므로, 사용자는 데이터 수집 중에 질의를 변경할 수 있다.

클라이언트와 서버의 동작 절차는 다음과 같다. 먼저, 클라이언트는 사용자로부터 데이터 스트림의 소스 정보를 입력 받는다. 그리고, 데이터 스트림의 스키마 정보를 입력 받아 데이터베이스에 저장한다(①). 두 번째로, 클라이언트는 사용자로부터 처리된 데이터를 출력할 데스티네이션 정보를 입력 받아 데이터베이스에 저장한다(②). 세 번째로, 데이터 스트림에 적용할 질의를 입력 받고, 서버는 이를 데이터베이스에 저장한다(③). 네 번째로, 사용자가 쿼리 필터링을 실행하면 서버는 스톰 클러스터에 명령어를 전달한다. 스톰은 데이터베이스에서 데이터 스키마와 질의를 읽어와 쿼리 필터링 토폴로지를 구성한다(④). 다섯 번째로, 클라이언트에서 데이터 입력을 실행하면 서버에서 데이터를 읽어 카프카로 입력하고, 스톰 토폴로지는 카프카로 들어온 데이터를 읽어 질의 필터링을 수행한다(⑤). 필터링된 데이터 스트림은 앞서 정의한 데스티네이션으로 출력한다.

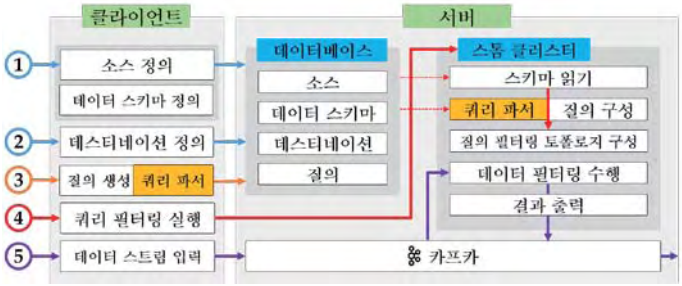

그림 2. 아파치 스톰 기반 질의 필터링 시스템 구조도.

3.1 질의 필터링 클라이언트

클라이언트 웹 UI에서는 먼저 사용자 인증 단계를 거친다. 이 단계에서는 계정 생성과 로그인을 할 수 있다. ID, 이름, 비밀번호 정보를 관리하고, 이를 통해 사용자를 식별한다. 사용자 인증 단계를 거친 후엔 각각의 메뉴를 통해 해당 사용자의 소스, 데스티네이션, 질의 필터링 토폴로지 정보를 보여준다.

소스 메뉴에서는 소스들의 이름, 생성 및 수정 날짜, 실행 상태 등을 확인할 수 있다. 또한, 새로운 소스를 생성하는 페이지로 이동할 수 있다. 소스 생성 페이지에서는 먼저 소스의 이름을 입력 받고, 사용자가 가진 소스 중 중복이 있는지 검사한다. 다음으로 소스의 타입을 지정한다. 소스의 타입이 카프카라면 카프카 클러스터의 주키퍼(Zookeeper) 접속 정보와 읽어올 토픽을 입력 받는다. 데이터베이스라면 데이터베이스의 접속 정보와 Query를 입력해야 한다. 그 외에도 사용자가 직접 정의할 수 있는 커스터마이징(customizing) 소스도 제공한다. 자바 인터페이스를 제공하여 사용자가 직접 시스템의 카프카로 데이터를 입력할 수 있다. 마지막으로 데이터의 스키마를 지정해야 한다. 이 단계에서는 데이터 칼럼의 이름과 데이터 타입을 입력한다. 데이터 타입은 `TEXT`, `NUMERIC`, `DATE`를 지원한다.

데스티네이션 메뉴에서는 데스티네이션들의 이름, 생성 및 수정 날짜, 실행 상태를 확인할 수 있다. 또한, 새로운 데스티네이션을 생성하는 페이지로 이동할 수 있다. 데스티네이션 생성은 소스 생성과 비슷한 단계를 거친다. 하지만, 소스의 데이터 스키마를 그대로 유지하므로 데이터 스키마를 정의하는 단계는 생략된다.

질의 생성 단계에서는 먼저 필터링을 적용할 소스를 선택한다. 다음으로, 필터링 하고자 하는 대상 칼럼을 선택하고, 연산자와 피연산자를 선택한다. 피연산자의 데이터 타입은 `TEXT`, `NUMERIC`, `TIMESTAMP`를 지원하며, 피연산자에 따라 선택할 수 있는 연산자의 종류는 표 1과 같다. 그리고, 연산자들은 `AND`, `OR` 같은 논리 연산자로 연결될 수 있다.

표 1. 피연산자에 따른 사용 가능한 연산자.

Data Type	Operator
`TEXT`	`equal not_equal in not_in`
`NUMERIC`	`equal not_equal greater less greater_or_equal less_or_equal`
`TIMESTAMP`	`greater_or_equal less_or_equal`

소스, 데스티네이션, 질의를 생성하면, 사용자는 질의 필터링과 데이터 스트림의 수집을 시작할 수 있다. 클라이언트의 질의 필터링 토폴로지 메뉴에서 쿼리 필터링 실행 명령을 보내면, 서버는 스톰 클러스터에 질의 필터링 토폴로지 구성 명령을 전달한다. 그리고, 소스 메뉴에서 데이터 스트림 수집 명령을 전달하면, 스톰 토폴로지는 데이터 스트림의 수집을 시작하고 질의 필터링을 적용한다.

3.2 질의 필터링 서버

서버는 데이터베이스, 카프카, 스톰 클러스터로 구성되며, 웹 UI에서 입력된 명령어를 처리하고 소스와 데스티네이션 정보를 사용하여 데이터를 읽거나 출력한다. 데이터베이스는 사용자가 클라이언트를 통해 입력한 사용자 정보, 소스 정보, 데이터 스키마 정보, 데스티네이션 정보, 그리고 질의를 저장한다.

스톰 클러스터는 사용자의 질의에 따라 질의 필터링 토폴로지를 구성하고 수행하는 역할을 한다. 먼저, 질의 필터링 수행 명령이 들어오면 데이터를 읽어올 소스 정보와 데이터의 스키마 정보를 데이터베이스에서 읽어온다. 또한, 사용자의 질의를 데이터베이스에서 읽은 뒤 쿼리 파서를 통해 질의를 구성한다. 사용자가 데이터 수집을 시작하면, 서버는 데이터베이스에 저장된 소스 정보를 이용하여 데이터를 수집한다. 수집된 데이터는 시스템의 카프카로 전달되고, 스톰은 카프카를 통해 데이터 스트림을 읽고 필터링을 수행한다. 처리된 데이터 스트림은 사용자는 미리 정의했던 데스티네이션으로 출력한다.

쿼리 파서 모듈은 웹 UI에서 생성된 JSON 형태의 질의를 데이터베이스에 저장하고, 스톰 토폴로지에서 데이터 스트림에 적용할 질의를 구성할 수 있도록 변환하는 역할을 한다. 쿼리 파서 모듈은 노드(Node), 연산자(Operator), 데이터베이스 어댑터(Database Adapter)로 구성되어 있으며, 이는 그림 3에 나타나 있다. 먼저, 노드는 피연산자와 연산자 정보를 관리한다. 이 때, 그림에서 `left`와 `right`는 피연산자로 숫자나 문자열 같은 값이거나 또 다른 노드가 될 수 있다. 노드의 메소드인 `parse`는 JSON 타입의 질의를 분석하여 시스템에서 사용하는 형태의 질의로 변환한다. 연산자는 노드의 문자열, 숫자, 날짜 데이터들에 대한 관계 연산자와 `AND`, `OR` 같은 연산자 간의 논리 연산을 관리한다. 그리고 노드의 `evaluate` 함수가 호출되면, 연산자의 `evaluate` 함수가 호출되어 입력된 데이터 스트림이 질의를 만족하는지 판단한다. 데이터베이스 어댑터는 노드와 연산자에 저장된 정보를 데이터베이스에 저장하는 역할을 한다. `addQuery` 메소드를 통해 데이터베이스에 노드와 연산자를 저장할 수 있으며, `getQuery` 메소드를 통해 저장된 정보를 다시 노드와 연산자로 구성할 수 있다.

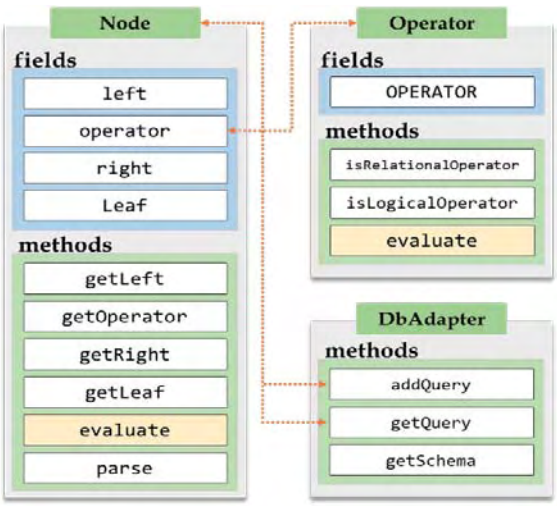

그림 3. 쿼리 파서의 클래스 구조.

4. 질의 필터링 시스템 구현 및 평가

본 절에서는 제안하는 아파치 스톰 기반 질의 필터링 시스템의 동작을 평가한다. 실험에 사용한 하드웨어 시스템 사양은 다음과 같다. 먼저, 서버는 Intel Xeon 2.4GHz 8 Core, 16GB RAM이며, 스톰 클러스터는 서버와 Intel Xeon 2.4GHz 6 Core, 16GB RAM 서버 여덟 대를 사용하여, 총 아홉 대의 서버를 EFM ipTIME T16000 네트워크 장비를 통해 분산 환경으로 구축하였다. 서버와 클라이언트는 Java와 Apache Tomcat 9.0을 기반으로 구현하였다. 테스트에 사용한 데이터 스트림은 트위터 데이터로 트위터 API[12]를 통해 실시간 입력 받아 사용한다. 데이터는 사용자 이름, 언어, 지역, 생성일, 리트윗 수, 내용의 여섯 가지 스키마로 구성되어 있다.

먼저, 트윗 데이터를 실시간으로 읽는 소스를 생성한다. 그림 4는 제안하는 필터링 시스템의 스키마 구성 화면이다. 그림에서 ⓐ부분은 데이터 칼럼의 이름을 입력하는 부분이다. ⓑ부분은 데이터 칼럼의 타입을 선택하는 부분으로, `TEXT`, `NUMERIC`, `DATE` 중 하나를 선택 가능하다. ⓒ버튼을 눌러 칼럼을 추가할 수 있고, ⓓ버튼으로 삭제할 수 있다. 또 드래그를 통해 칼럼의 순서를 바꿀 수 있다(ⓔ). 트윗 데이터의 스키마에 맞게 칼럼과 타입을 입력한다. 필터링된 데이터는 시스템의 카프카에서 확인할 수 있도록 설정한다.

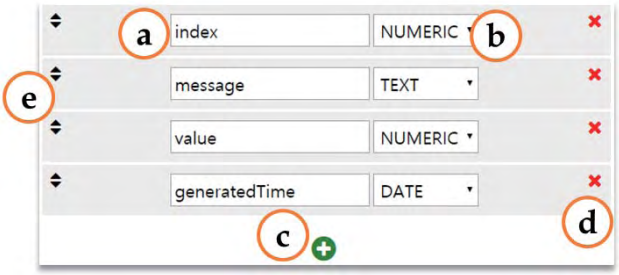

그림 4. 데이터 스키마 정의 페이지.

그림 5는 데이터 스트림에 적용할 질의를 구성하는 부분이다. ⓐ부분은 스키마 정의 단계에서 입력한 데이터 칼럼을 피연산자로 선택할 수 있다. 다음으로, ⓑ부분에서 데이터 타입에 따라 연산자를 선택하고, ⓒ에서 질의 조건을 결정할 수 있다. 또, ⓓ에서 질의 조건을 추가할 수도 있다. 테스트에서는 'language equal en'으로 질의를 생성한다. 즉, 트윗 중 언어가 영어인 데이터만을 저장하고, 다른 언어의 트윗은 저장하지 않는다.

그림 5. 질의 생성 페이지.

질의 생성을 마치면, 그림 6처럼 질의 필터링 토폴로지 메뉴에서 확인할 수 있다. 토폴로지의 이름과, 생성 날짜, 동작 상태 등을 확인할 수 있으며, ⓐ버튼을 눌러 해당 질의 필터링 토폴로지를 실행할 수 있다. 다음으로, 소스 페이지에서 미리 정의한 트윗 데이터 수집을 실행한다.

그림 6. 질의 필터링 토폴로지 메뉴.

그림 7은 실시간 트위터 스트림에 질의 필터링 적용한 결과의 일부이다. 질의 생성 시 언어 칼럼이 영어인 데이터만 수집하도록 설정하였고, 그 결과 그림 7에서 언어가 영어인 데이터만 수집된 것을 확인할 수 있다.

그림 7. 질의 필터링 결과.

5. 결론 및 향후 연구

본 논문에서는 데이터 스트림 대상의 필터링 시스템을 아파치 스톰 기반으로 설계하고 구현하였다. 제안하는 필터링 시스템은 웹 UI를 통해 데이터 스트림에 질의 필터링을 적용할 수 있다. 또한, 데이터 스트림의 수집, 처리, 출력의 작업을 별도의 모듈로 구현함으로써, 입력 데이터의 구조나 질의의 변경에도 별도의 코드 수정 없이 시스템을 사용할 수 있다. 실험 결과, 제안하는 필터링 시스템은 사용자가 입력한 데이터의 구조와 질의에 따라 데이터 스트림을 수집하고 필터링하는 것을 확인하였다. 향후 연구에서는 CSV 이외에 다양한 구조의 데이터를 지원하도록 확장할 예정이다.

참고문헌

[1] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. Byers, "Big Data: The Next Frontier for Innovation, Competition, and Productivity," Technical Report, McKinsey Global Institute, 2011.

[2] Apache Storm, http://storm.apache.org/.

[3] Toshniwal, et al., "Storm@Twitter," In *Proc. of the Int'l Conf. on Management of Data*, ACM SIGMOD, Utah, USA, pp. 147-156, June 2014.

[4] Apache Spark Streaming, https://spark.apache.org/streaming/.

[5] Apache Kafka, http://kafka.apache.org/.

[6] J. Kreps, N. Narkhede, and R. Jun, "Kafka: a Distributed Messaging System for Log Processing," In *Proc. of the NetDB*, Athens, Greece, pp. 1-7, June 2011.

[7] Storm and Kafka Together: A Real-Time Data Refinery, https://hortonworks.com/blog/storm-kafka-together-real-time-data-refinery/.

[8] J. Leskovec, A. Rajaraman, and J. Ullman, Mining of Massive Datasets, Cambridge University Press, 2015.

[9] E. Alpaydin, Introduction to Machine Learning, MitPr, 2nd ed. 2010.

[10] jQuery QueryBuilder, https://querybuilder.js.org/.

[11] Reynold Cheng, et al., "Filtering Data Streams for Entity-Based Continuous Queries," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 22, Issue 2, pp. 234-248, Feb. 2010.

[12] Twitter API, https://developer.twitter.com/.