Journal of The Korea Society of Computer and Information
Vol. 23 No. 11, pp. 17-24, November 2018
www.ksci.re.kr
https://doi.org/10.9708/jksci.2018.23.11.017

Efficient *K*-Anonymization Implementation with Apache Spark

Tae-Su Kim*, Jong Wook Kim*

Abstract

Today, we are living in the era of data and information. With the advent of Internet of Things (IoT), the popularity of social networking sites, and the development of mobile devices, a large amount of data is being produced in diverse areas. The collection of such data generated in various area is called big data. As the importance of big data grows, there has been a growing need to share big data containing information regarding an individual entity. As big data contains sensitive information about individuals, directly releasing it for public use may violate existing privacy requirements. Thus, privacy-preserving data publishing (PPDP) has been actively studied to share big data containing personal information for public use, while preserving the privacy of the individual. *K*-anonymity, which is the most popular method in the area of PPDP, transforms each record in a table such that at least *k* records have the same values for the given quasi-identifier attributes, and thus each record is indistinguishable from other records in the same class. As the size of big data continuously getting larger, there is a growing demand for the method which can efficiently anonymize vast amount of dta. Thus, in this paper, we develop an efficient *k*-anonymity method by using Spark distributed framework. Experimental results show that, through the developed method, significant gains in processing time can be achieved.

▸Keyword: *K*-anonymity, Spark, Hadoop, Distributed system, Data privacy

I. Introduction

오늘날 세계는 데이터 중심으로 돌아가고 있다. 다양한 분야에서 데이터가 끊임없이 생산되는 시대이다. 인터넷의 발전, 모바일 기기의 발전, 사물인터넷 기술의 등장, SNS의 활성화 등으로 하루에도 수많은 데이터들이 쏟아지고 있다. 과거에 비해 데이터를 통해서 작업을 하는 일이 많아지고 중요해지고 있다. 이러한 데이터들은 엄청난 가치를 지니고 있다. 빅데이터라는 단어가 나오면서 방대한 데이터들을 관리하고 활용하는 기술이 대두되기 시작했다.

빅데이터는 여러 분야에서 활용도가 급증하고 있다. 빅데이터의 시장가치는 연간 계속해서 증가하고 있는 추세이며 많은 기업들이 빅데이터를 경영에 활용하고 있다. 빅데이터는 정치, 경제, 문화, 과학기술, 기상정보 등 여러 분야에서 활용되고 있다. 빅데이터의 지속적인 관측과 분석으로 미래의 사회를 좀 더 정확하게 예측 할 수 있고, 모든 분야에서 가치 있는 정보를 창출해 낼 수 있다.

미래의 빅데이터는 현재의 빅데이터의 규모, 중요성보다 더욱 커질 것이다. 빅데이터의 중요성이 커지는 만큼 빅데이터를 공유 및 배포에 대한 관심이 높아지고 있다. 그러나 일반적으로 빅데이터 개인의 민감한 정보를 포함하고 있으므로, 빅데이터를 공유 및 배포하는 과정에서 제 3자(공격자)가 악용하면 수많은 개인정보가 유출되고 유출된 개인 정보는 큰 피해를 불러올 수 있다. 빅데이터 공유 및 배포 시 이러한 개인정보 유출 문제를 해결하기 위한 주된 기술로 데이터 익명화 기술이 사용되고 있다.

데이터 익명화는 제 3자가 데이터를 봤을 때 특정 개인을 식별하지 못하게 데이터를 익명화하는 작업을 말한다. 익명화된

• First Author: Tae-Su Kim, Corresponding Author: Jong Wook Kim
*Tae-Su Kim (akvks456@gmail.com), Dept. of Computer Science, Sangmyung University
*Jong Wook Kim (jkim@smu.ac.kr), Dept. of Computer Science, Sangmyung University
• Received: 2018. 08. 31, Revised: 2018. 10. 04, Accepted: 2018. 10. 22.
• This research was supported by a 2017 Research Grant from Sangmyung University.

데이터를 공유 및 배포함으로써 개인을 식별할 수 없기 때문에 익명화 기술은 빅데이터 배포에 사용되는 가장 기본적인 기법이다. 현재까지 다양한 익명화 기술이 연구되어 왔다.[1,2,3,4] 가장 대표적인 익명화 기술로 k-익명화 기법이 있다.[5,6,7,8] 대부분의 익명화 기법들은 데이터를 크게 3개의 구성요소로 구분 한다. 데이터에서 한 개인의 정보를 알 수 있는 정보인 '식별자 (identifier)', 하나의 정보만으로는 특정한 개인을 식별할 수 없지만 하나 이상의 정보와 제 3자의 배경지식 혹은 외부데이터와 결합하면 특정 개인을 식별할 수 있는 '준식별자 (quasi-identifier)', 사용자의 민감한 정보에 해당하는 '민감 속성 (sensitive attribute)'가 있다. k-익명화 기법은 먼저 식별자를 제거하고, 준식별자를 일정한 범주로 일반화한다. 일반화된 준식별자는 일정한 범주로 동질 클래스(equivalence class)를 형성하게 된다. 각각의 동질 클래스에 속하는 레코드의 수가 k개 이상을 만족하도록 준식별자를 일반화 한다. 이를 통해 제 3자가 배경지식과 외부 데이터를 이용하더라도 각각의 동질 클래스에 적어도 k개의 데이터가 존재하기 때문에, 특정한 개인을 식별할 수 없게 된다. 따라서 k-익명화를 통해 안전하게 데이터를 공유 및 배포할 수 있다.

최근 들어 빅데이터의 규모가 점점 커짐에 따라 분산 처리를 이용하여 데이터를 효율적으로 처리하기 위한 기술 개발에 대한 관심이 높아지고 있다. 빅데이터 분산 처리를 지원하는 기술 중 대표적인 기술로 하둡(Apache Hadoop)과 스파크(Apache Spark)가 있다.[10, 12] 하둡은 빅데이터를 처리할 수 있는 분산 응용 프로그램을 지원한다. 작업을 n개의 클러스터로 분산한 후, 하둡 맵리듀스(MapReduce) 작업을 수행 한다. 하둡 맵리듀스는 구글에서 빅데이터를 분산 환경에서 처리하기 위해 제작한 소프트웨어 프레임워크이다. 스파크는 하둡과 비슷한 기능을 하는 범용 분산 플랫폼이다. 하둡의 맵리듀스 과정은 스파크의 데이터 처리방식보다 느리다. 스파크의 데이터 처리방식은 전체 데이터셋을 메모리에서 처리하기 때문에 속도가 매우 빠르다. 하둡은 스파크와 달리 클러스터의 하드디스크를 기반으로 데이터를 처리하기 때문에 데이터의 크기가 커진다면 데이터를 로드하는 작업에 있어서 효율이 떨어진다. 반면 스파크는 인 메모리 기반으로 데이터를 메모리에 로드하여 작업을 처리하기 때문에 작업 속도 면에서 하둡보다 우수하다.

본 논문은 스파크 분산 환경에서 k-익명화 기법을 개발한다. 기존의 k-익명화 기법과 달리 본 논문에서 개발한 k-익명화 기법은 스파크 분산 환경을 이용하여 대용량의 데이터를 빠르게 익명화 할 수 있다는 장점이 있다. 또한, 실제 분산환경에서 실데이터를 이용하여 본 논문에서 개발한 스파크 기반 k-익명화 기법의 효율성을 검증한다. 본 논문은 다음과 같이 구성되어 있다. 2장에서 본 논문의 배경지식에 대해 설명하고 3장에서는 스파크를 이용한 분산 처리와 k-익명화에 대해 설명을 한다. 4장에서 본 논문에서 제안하는 기법의 성능평가를 수행한 후, 5장에서 결론을 맺는다.

II. Background

1. K-anonymity

한 개의 데이터는 수많은 정보들을 담고 있다. 이 데이터 속에서 특정한 개인을 식별할 수 있는 속성을 '식별자' 라고 한다. 식별자의 예로는 전화 번호, 이름, 주민등록번호, 등이 있다. 제 3자(공격자)가 데이터의 식별자를 알고 있다면 개인 정보가 유출된 것이다. 그러므로 데이터 공유 시에는 식별자들을 삭제하고 공유하는 것이 원칙이다. 하지만 데이터 내에서 식별자를 제거해도, 외부 데이터, 공격자의 배경지식 등과 결합해 특정 개인을 식별할 수 있는 속성이 있다. 이렇게 단일 데이터만으로는 개인을 식별할 수 없지만 다른 데이터들과의 결합을 통해 개인을 식별할 가능성이 있는 속성을 '준식별자'라고 한다. 따라서 데이터 공유에 있어 식별자 제거와 함께 준식별자를 비식별화 할 필요가 있다. 가장 대표적인 프라이버시 모델인 k-익명화 기법은 준식별자에 대하여 동일한 속성 값을 가지는 레코드들의 집합인 동질 클래스의 크기를 k 이상으로 요구함으로, 특정 레코드를 동질 클래스내의 다른 (k-1)개의 레코드들과 구별할 수 없게 한다. [5,7]. 가령, 표 1은 원본 테이블과 2-익명화 테이블을 나타내며, 준식별자는 'Age'와 'Gender' 에 해당된다. 2-익명화 테이블에는 2개의 동질 클래스(2~3번, 6번 레코드들로 구성된 동질 클래스, 1번, 4~5번 레코드들로 구성된 동질 클래스)가 존재한다. 익명화된 테이블에서 각각의 레코드는 동질 클래스 내의 다른 레코드들과 구분이 되지 않는다. 가령, 1번 레코드는 2-익명화 테이블에서 다른 두 개의 레코드들(RID=4,5)과 구별할 수 없으므로, 익명성이 보장된다. k-익명화에서 데이터를 변형하는 대표적인 방법으로는 일반화(generalization) 기법이 사용된다.

Table 1. Original table and 2-Anonymized table

Original Table			
RID	Age	Gender	Disease
1	31	M(1)	Diabetes
2	21	M(1)	Anemia
3	26	F(0)	Pneumonia
4	36	F(0)	Anemia
5	34	M(1)	Diabetes
6	25	F(0)	Pneumonia
2-Anonymized Table			
RID	Age	Gender	Disease
1	30~39	*(0~1)	Diabetes
2	20~29	*(0~1)	Anemia
3	20~29	*(0~1)	Pneumonia
4	30~39	*(0~1)	Anemia
5	30~39	*(0~1)	Diabetes
6	20~29	*(0~1)	Pneumonia

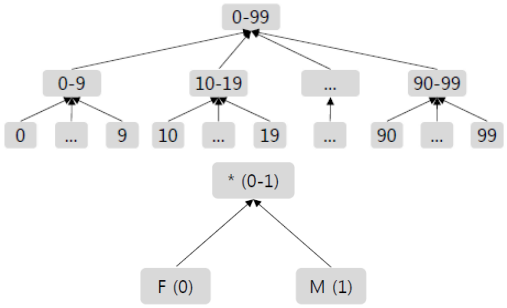

Fig. 1. Example taxonomy trees of Age and Gender

일반화 기법은 원래의 데이터를 일반화 규칙에 따라 일반화 데이터로 대체함으로써 데이터를 익명화하는 방법이다[5,7,9]. 일반화 규칙은 일반화 방법에 적용되는 규칙으로써, 각 속성의 범주 트리(taxonomy tree)를 토대로 만들어진다. 범주 트리는 계층적 트리로서, 상위 단계일수록 높은 일반화 수준을 나타낸다. 예를 들어, 그림 1은 'Age'와 'Gender' 속성에 대한 범주 트리다. 그림 1 에서 보는 것과 같이 'Age'에 대한 범주 트리는 3단계의 일반화 수준을 가지고 있고, 'Gender'에 대한 범주 트리는 2단계의 일반화 수준을 가지고 있다. 그림에서 보듯이 낮은 일반화 수준일수록 원래의 값과 가깝고, 높은 일반화 수준일수록 일반화된 값의 범위가 넓다. 그러므로 일반화 수준이 낮아지면 데이터의 활용도가 높아지지만 개인정보 유출의 위험도 또한 증가한다. 반대로 일반화 수준이 높아지면 개인정보 유출의 위험도가 낮아지지만 데이터의 활용도 또한 낮아진다. 그래서 전역 일반화 방법에서의 핵심은 적절한 범주 트리의 형태와 그 형태에 따른 적합한 일반화 규칙을 구성하는 것에 있다. 또한, 그림 1의 'Gender'에 해당하는 범주 트리에서 보여 지듯이, 범주형 자료(categorical data)는 수치적 자료(numerical data)로 표현 가능하다.

2. Apache Hadoop

하둡은 빅데이터를 저장, 처리할 수 있는 소프트웨어 프레임워크이다.[10] 하둡은 로컬(local mode), 스탠드 얼론(standalone mode), 분산(distributed mode)의 총 3가지 모드를 설정할 수 있다. HDFS (Hadoop Distributed File System)는 데이터를 여러 대의 서버에 분산 저장하고 저장한 데이터를 빠르고 효율적으로 처리할 수 있게 해주는 파일 시스템이다. HDFS는 블록 구조의 파일시스템이므로 데이터가 일정 블록크기만큼 분산 저장된다. 블록 크기의 기본 값은 128MB로 설정되어있지만 사용자가 크기를 변경할 수 있다. HDFS는 네임노드와 데이터노드의 마스터/슬레이브(master/slave) 구조로 구현된다. 네임노드는 기능은 HDFS 내부의 블록 관리, 데이터노드 모니터링, 메타데이터 관리 및 처리 등이 있다. 데이터 노드는 HDFS에 저장하는 파일을 데이터 노드를 가지고 있는 여러 개의 노드의 로컬 디스크에 유지한다. 그림 2는 네임노드와 데이터노드의 구조를 나타낸 그림이다.

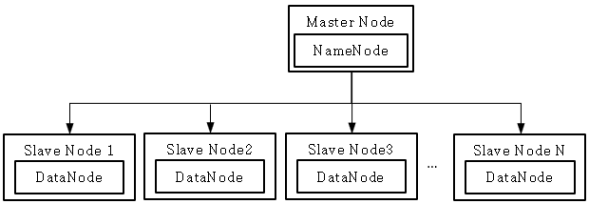

Fig. 2. HDFS Architecture

하둡은 맵리듀스 기반으로 데이터 처리를 진행한다[11]. 하둡 맵리듀스란 대용량 데이터를 분산 병렬 컴퓨팅 환경에서 처리하기 위해 만든 소프트웨어 프레임워크이다. 입력과 출력으로 키-값(key-value)의 쌍을 값으로 가지고 있다. 하둡 맵리듀스는 맵(Map)과 리듀스(Reduce) 과정으로 구성되어 있다. 맵 과정은 초기 데이터를 키-값의 쌍으로 데이터를 분류하고, 리듀스 과정은 맵의 결과인 키-값의 데이터를 프로그래머가 원하는 결과에 맞게 추려내는 과정이다.

여러 클러스터에서 병렬로 처리하기 위해 하둡은 HDFS와 얀(Yarn)을 이용한다. 얀은 하둡 2.0부터 나온 클러스터 관리자 이다. 얀은 기존 하둡 맵리듀스의 단점인 클러스터 전체 사용률을 크게 보완하였고 리소스 관리와 스케줄링, 모니터링을 분리함으로써 확장성의 범위를 크게 확대했다. 얀은 하둡 맵리듀스의 기능을 구현할 수 있을 뿐만 아니라 다른 분산 처리 프레임워크(예, 스파크)와도 호환이 잘 된다. 그림 3과 같이 얀은 자원 관리를 위해 리소스매니저(ResourceManager)와 노드매니저(NodeManager)라는 컴포넌트로 구성되어 있다. 리소스매니저는 전체 클러스터에서 사용가능한 모든 시스템 자원을 관리하며, 마스터 서버에서 실행된다. 노드매니저는 하둡 맵리듀스 작업을 담당하며, 각 클러스터에서 실행된다. 노드매니저는 컨테이너(Container)를 실행하고 컨테이너들을 모니터링 한다. 컨테이너는 노드매니저를 실행하는 서버의 시스템 자원에 대한 정보를 가지고 있다. 노드매니저는 컨테이너 단위로 애플리케이션을 실행하고 스케줄링 한다. 노드매니저에는 애플리케이션 마스터라는 서버가 존재한다. 사용자가 얀에 애플리케이션 실행을 요청하면 얀은 하나의 노드매니저에 하나의 애플리케이션 마스터를 할당한다. 애플리케이션 마스터는 애플리케이션 실행에 필요한 자원을 스케줄링하고, 노드매니저에 해당 애플리케이션에 필요한 컨테이너를 실행하도록 요청한다.

하둡은 초기 설치할 때 각 서버에 맞게 설정 값들을 교정할 수 있다. 메모리를 사용하는 각 방법에서 메모리의 양을 조절할 수 있고 사용자 편의에 맞게 포트와 데이터노드를 설정할 수 있어 최적의 분산처리 환경을 만들 수 있다.

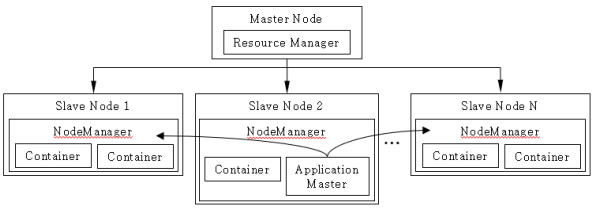

Fig. 3. Hadoop Yarn Architecture

3. Apache Spark

하둡 맵리듀스는 디스크 기반이기 때문에 디스크 크기가 크면 많은 양의 데이터를 처리할 수 있다는 장점이 있지만, 속도가 느리다는 단점이 있다. 최근 들어 이러한 문제점을 개선한 인 메모리 기반의 분산 시스템이 나오기 시작했다. 아파치 스파크는 범용적이고 빠른 속도로 작업을 수행할 수 있도록 설계한 인 메모리 기반 클러스터용 연산 플랫폼이다.[12] 기존 분산시스템에서 동시에 실행하지 못하는 배치 애플리케이션, 반복 알고리즘, 대화형 쿼리 등 많은 작업 타입을 스파크는 동시에 작업할 수 있다. 스파크의 장점은 속도, 다양한 시스템과의 호환, 인 메모리 등이다. 단계적으로 데이터를 처리하는 방식인 하둡 맵리듀스와 달리 스파크는 데이터를 RDD(Resilient Distributed Dataset)로 만들고 한번에 처리한다. 하둡 맵리듀스보다 처리하는 과정이 줄어들기 때문에 처리 속도가 더 빠르다.

스파크는 RDD를 이용하여 분산 환경에서 외부 데이터를 메모리에 저장한다. RDD는 분산 환경에서 작업할 수 있도록 여러 개의 파티션으로 나뉜다. 스파크는 분산 모드에서 하나의 스파크 드라이버와 여러 개의 워커 노드로 구성되는 마스터/슬레이브 구조를 사용한다. 워커 노드 내부에는 분산된 작업을 수행하는 익스큐터(Executor)들이 존재한다. 스파크 애플리케이션은 스파크 드라이버와 익스큐터들의 집합이다. n대의 서버 전부 워커 노드의 역할을 하며 n대의 서버 중 1대의 서버는 워커 노드들을 관리하는 스파크 드라이버 역할을 한다. 드라이버가 된 서버는 SparkContext(스파크 애플리케이션 실행과 관련된 정보를 갖고 있는 객체)를 생성하고 작업을 할 RDD를 생성한다. 생성한 RDD로 트랜스포메이션과 액션 연산을 실행하는 코드를 실행한다.

드라이버는 2가지 기능을 한다. 그림 4와 같이 첫 번째로 생성한 RDD를 기반으로 RDD 연산 관계를 바탕으로 DAG(RDD의 연산 관계에 대해 논리적인 지향성 비순환 그래프)를 생성하고 이것을 물리적 실행 계획으로 변환한다. DAG를 작업의 기본 단위인 태스크(task)들로 구성된 단계(stage)로 변환한다. 각 워커 노드에게 태스크를 할당하기 위해 각 단계는 태스크로 분할된다. 태스크는 각 클러스터에게 전달되고 클러스터는 태스크의 작업을 수행한다. 두 번째로 워커 노드 내부에 있는 익스큐터들의 개별 작업을 스케줄링 한다. 익스큐터들은 드라이버로부터 받은 태스크의 작업을 수행하고 드라이버에게 결과를 보낸다. 결과를 받은 드라이버 프로그램은 최종 산출물인 새로운 RDD를 저장하거나 출력하고 종료하고 동시에 스파크 애플리케이션이 종료된다.

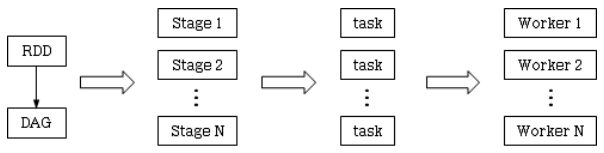

Fig. 4. Spark Workflow

스파크 애플리케이션을 실행하기 위해서는 스파크 클러터 관리자(Spark Cluster Manager)가 필요하다. 스파크 클러스터 관리자는 애플리케이션을 실행하는데 필요한 자원을 할당한다. 대표적으로 스탠드얼론, 메소스(Apache Mesos), 얀이 있다. 얀은 스파크 분산 환경에서 2가지 클러스터(cluster)모드와 클라이언트(client) 모드가 있다. 클러스터 모드는 스파크 드라이버가 얀 컨테이너에서 실행중인 스파크 애플리케이션 관리자에서 실행된다. 동일한 프로세스가 애플리케이션을 구동하고 필요한 자원을 요청을 동시에 수행할 수 있다. 클러스터 모드는 생산 작업에 유리하다. 클라이언트 모드는 스파크 드라이버가 작업을 요청한 클라이언트 머신에서 실행된다. 애플리케이션 관리자는 얀에서 익스큐터 컨테이너를 요청하는 한 가지 일만 한다.

III. K-Anonymity on Spark

본 연구에서는 스파크 분산 환경을 이용하여 효율적으로 대용량 데이터에 대한 k-익명화 작업을 수행하기 위한 기법을 제시한다. 본 논문에서 제안하는 스파크 분산처리를 이용한 k-익명화 알고리즘은 다음과 같이 구성 된다 (그림 5).

(1) k-익명화 작업의 대상인 원본 데이터들을 HDFS에 로드한다.
(2) k-익명화 기법의 일반화를 수행하기 위하여 범주 트리를 생성한 후, 이를 이용하여 일반화 격자를 생성한다.
(3) HDFS에 있는 원본 데이터를 RDD로 변환한다.
(4) RDD를 메모리에 로드하고 k-익명화 작업을 수행한다.

본 절에서는 위의 과정을 구체적으로 설명한다.

```
//Step 1: Load Files to HDFS
1.      Load_Data_to_Memory(data);

Main
Input: k-value, data, partition num, attr
Output: k-value, node, k-Anonymized table

//Step 2: Create Taxonomy Tree and
          Generaliztion Lattice Tree
2.      Make_Taxonomy Tree(attr);
3.      Make_Generalization_Lattice(t_tree);

//Step 3: Make RDD and Partition, Cache
4.      Make_RDD_From_HDFS(data);
5.      RDD_Repartition(partition num);
6.      Cache();

//Step 4: K-Anonymity (Map & Reduce)
7.      k_check = false;
8.      while(!k_check)
9.          node = next Generalization Lattice;
10.          result = MapReduce(node);
11.          k_check = Check_k_value(result);
12.     end while;
13.     Save_Output_to_HDFS(path);
```

Fig. 5. The Pseudocode for k-Anonymity in Spark Distributed Environment

3.1 *Step 1: Load Data to HDFS*

첫 번째 단계에서는 스파크 분산처리에 HDFS를 이용하기 위해 *k*-익명화 작업의 대상인 원본 데이터들을 HDFS에 로드한다. 원본 데이터들 HDFS에 로드하면 HDFS의 블록 크기만큼 자동으로 원본 데이터가 분할된다. 그림 6에서 보이듯이, 분할 된 데이터는 데이터 노드로 지정된 노드에 분산 저장된다.

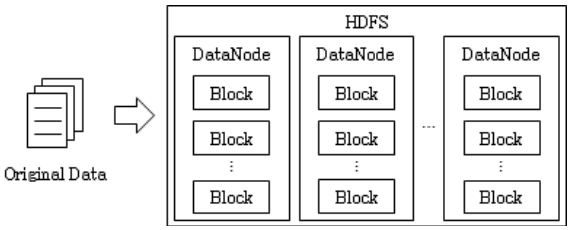

Fig. 6. Load Files to HDFS

3.2. *Step 2: Create Taxonomy Tree and Generalization Lattice Tree*

데이터가 HDFS에 저장되면 1개의 노드가 드라이버가 되고 사용자가 정의한 Main 함수를 실행한다. 알고리즘의 두 번째 단계에서는 우선 범주 트리의 속성이 메모리에 로드되고 속성에 맞게 일정 범위로 나누어진 레벨로 범주 트리가 생성된다. 그리고 생성된 범주 트리를 토대로 일반화 격자(Generalization Lattice)를 만든다. 일반화 격자는 각 속성의 범주 레벨 최하위부터 최상위까지의 조합으로 생성된다. 생성된 일반화 격자는 가장 하위 단계부터 일반화에 사용된다. 일반화 격자 트리는 메모리에 저장된다.

그림 7은 그림 1의 'Age'와 'Gender' 속성에 해당하는 범주 트리가 주어졌을 때, 일반화 격자 트리를 나타낸 그림이다. 여기서, A와 G는 'Age'와 'Gender' 속성을 나타내면, 옆의 숫자는 범주 트리의 레벨을 나타낸다. 즉 'A0'은 'Age' 속성의 단말 노드들에 해당하고, 'A2'는 'Age' 속성의 루트 노드에 해당한다. 그림 7의 격자 트리를 기준으로 익명화를 수행하면서 최하위 노드부터 순서대로 진행한다. 가령 그림 7의 일반화 격자 트리의 경우, [A0,G0], [A1,G0], [A0,G1], …, [A2,G1] 순으로 익명화가 진행된다.

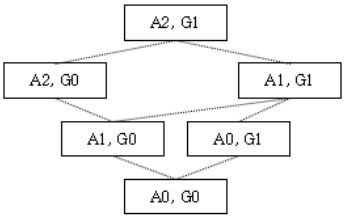

Fig. 7. Generalization Lattice Tree

3.3. *Step 3: Make RDD and Partition, Cache*

알고리즘의 세 번째 단계에서는 그림 8에서 보이듯이, HDFS에 있는 원본 데이터를 RDD로 변환한다. 만약 원본 데이터가 로컬 시스템에 존재하는 경우 각 워커노드 모두 같은 경로에 원본파일이 존재해야 한다. 변환하기 위해 sc(SparkContext)객체를 생성한다. sc객체가 만들어 지면 스파크 분산처리 및 RDD의 생성과 연산 등이 가능해진다. sc객체를 이용해 원본데이터를 RDD로 생성하고, 스파크 명령어에서 입력 받은 파티션 수만큼 RDD를 나눈다. 만약 파티션 수가 0이라면 RDD는 HDFS에서 블록 크기만큼 파티션된 개수를 유지한다. 파티션 과정을 마친 RDD는 더 빠른 분산 처리를 위해 메모리에 올려진다. 메모리에 올리는 방법은 직렬화 여부와 데이터 크기가 메모리 크기보다 클 경우 디스크 저장 여부에 따라 달라진다.

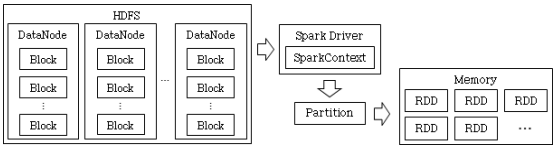

Fig. 8. Make RDD and Partition, Cache

3.4. *Step4: Spark-MapReduce for K*-Anonymization

그림 9는 만들어진 RDD를 스파크를 이용한 *k*-익명화의 맵 과정과 리듀스과정을 수행하고 k값 만족 여부를 측정하는 알고리즘의 전체적인 그림에 해당한다.

3.4.1 Spark-Map for *K*-Anonymization

스파크를 이용한 맵을 진행하기 위해 *Step2* 에서 만든 일반

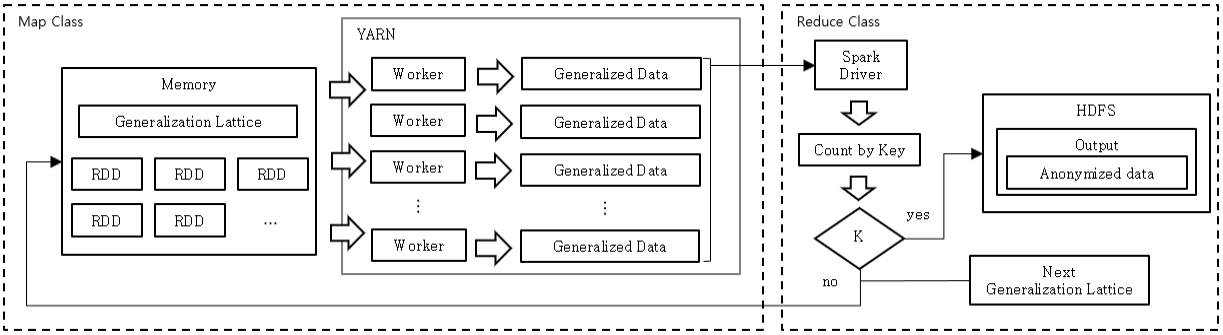

Fig. 9. K-Anonymity on Spark Map & Reduce

화 격자의 최하위 노드(각 속성의 범주 레벨이 0인 격자)와 *Step3* 에서 RDD로 바꾼 원본 데이터를 맵 함수의 파라미터로 입력한다. 맵 단계에서는 파라미터로 입력 받은 원본 데이터와 일반화 격자에 맞게 일반화하는 작업이 태스크로 만들어진다. 또한, 얀을 통해 태스크들이 각 워커들에게 전달되고, 생성된 익스큐터들이 태스크를 수행한다. 준식별자들은 선택된 일반화 격자에 맞춰서 일정한 범주로 값이 일반화된다. RDD 연산은 데이터를 일반화하기 위해 반복문을 추가적으로 넣지 않아도 전체 데이터셋을 한번에 처리할 수 있으므로 RDD내의 모든 라인을 매핑 후 일반화한다. 일반화된 데이터는 키-값의 형태로 저장된다. 키에는 일반화된 준식별자 데이터가 설정되고 값에는 일반화된 데이터의 민감 정보가 설정된다. 맵 과정이 완료되면 {*일반화된 클래스-민감정보*} 의 키-값 형태로 새로운 RDD를 만든다. 각 익스큐터의 맵 과정이 끝나면 리듀스 과정으로 넘어간다. 그림 10은 맵의 과정을 설명한 수도코드이다.

```
Map
Input: RDD of Original Table, node of Generalization Lattice
Output: {Generalize_Data, sq(Sensitive Information)}
1. Map_RDD(original data){
2.      q_line = line's quasi-identifiers;
3.      trans_line = Generalize_Data(q_line, node);
4.      write(trans_line,sq); //Output by JavaPairRDD
5. };
```

Fig. 10. The Pseudocode of Map for k-Anonymity

3.4.2 Spark-Reduce for K-Anonymization

맵 과정이 끝나면 스파크 드라이버가 스파크를 이용한 *k*-익명화의 리듀스를 수행하기 위해 모든 익스큐터의 맵 결과 값을 받는다. 익스큐터로부터 받은 일반화된 키-값을 토대로 리듀스 과정을 진행한다. 키가 같은 값을 묶어서 새로운 키-값을 만든다. 그 결과 키에는 동질클래스, 값에는 동질클래스를 갖는 민감정보의 개수가 저장된다. 그림 11은 리듀스 과정을 수도코드로 설명한 그림이다.

```
Reduce
Input: Output of Map(javapairRDD)
Output: Map<String, Long>= {equivalence class, num}
1. Reduce_RDD(map_output){
2.      Count_By_Key(javapairRDD);
3.      Write_to_RDD(equivalence class, num);
4. };
```

Fig. 11. The Pseudocode of Reduce for k-Anonymity

3.4.3 Check termination condition

리듀스 작업이 끝나면 스파크 애플리케이션을 실행할 때 명령 줄의 파라미터로 보낸 k값과 리듀스 결과 값의 동질클래스의 민감정보 개수를 비교연산 한다. k값 보다 작은 값이 존재하면 *k*-익명화를 만족하지 않기 때문에 즉시 비교 연산을 멈추고 현재 일반화 격자를 다음 단계의 격자로 바꾸고 반복문 초기로 돌아가 다시 맵 과정을 수행한다. 이 때, 원본 데이터를 디스크에서 다시 불러오는 것이 아닌 메모리에 로드 된 원본 데이터를 기반으로 다시 *k*-익명화를 수행한다.

리듀스의 결과 값의 모든 동질클래스의 민감정보 개수가 k값보다 같거나 크면 *k*-익명화의 조건을 만족하므로 반복문을 종료하고 사용자에게 작업 완료를 알린다. 이 때, 최종 결과로 k값, 익명화에 사용된 일반화 격자 노드 값, 익명화된 데이터가 출력된다. 익명화된 데이터는 사용자 편의에 따라 HDFS에 저장되거나 자신의 로컬 디스크에 저장 가능하다.

스파크를 이용한 *k*-익명화 맵리듀스 과정이 끝나면 메모리에 로드했던 원본 데이터 RDD는 메모리에서 삭제된다. 익명화에 사용된 자원들은 반환되며 최종적으로 스파크 애플리케이션이 종료된다.

IV. Experiment

본 절에서는 논문에서 제안한 기법의 성능 평가를 보건의료 데이터[13]를 사용하여 수행한다. 본 실험에서는 보건의료 데이터로부터 'Age', 'Sex', 'Location', 'Surgery', 'Length', 'Disease' 속성을 추출하여 원본 테이블을 생성하였다. 이때, 'Disease' 속성은 민감한 속성(sensitive attribute)에 해당하며, 그 이외의 속성들은 준식별자에 해당한다. 본 논문의 실험에서 사용한 데이터의 경우 많은 중복된 데이터를 포함하고 있기 때문에, k 값을 너무 작게 설정하면, 이미 데이터가 익명화 된 상태에 해당한다. 그러므로 본 논문에서는 이러한 데이터의 특성을 고려하여 k 값을 100, 500, 1000으로 설정하였다. 또한 *k*-익명화 알고리즘으로는 비분산 환경에서 실행되는 비분산 알고리즘(Non-distributed *k*-anonymity)과 본 논문에서 제안하는 스파크 분산 환경에서 실행되는 알고리즘(Distributed *k*-anonymity)를 이용하여 비교 성능 평가를 수행하였다.

본 실험에서는 스파크 분산 환경에서의 실험을 수행하기 위해 3대의 서버로 구성된 클러스터를 사용하였다. 표 2는 실험에 사용한 서버 사양과 스파크 옵션 값을 나타낸다.

Table. 2. Server Specs and Spark Options used in the experiments

Server Specs	HDD size	4 TB
	Memory size	64 GB
	CPU	4
	CPU cores	8
Spark Options	Executor-cores	4
	Executor-memory	10 GB
	Driver-memory	10 GB
	Java heap max	55 GB

1. Experimental Results

그림 12는 원본 테이블의 크기 변화에 따른 *k*-익명화를 수

행 시간을 비교한 실험 결과이다. 이 실험에서 k 값은 500으로 고정하였고, 원본 테이블의 크기는 0.1GB, 0.5GB, 1GB로 변화를 주었다. 또한 비분산 환경을 이용한 경우 JVM 메모리를 50GB로 할당하였다. 그림에서 보듯이 원본 테이블의 크기가 증가할수록 *k*-익명화 수행 시간이 증가함을 알 수 있다. 특히, 원본 테이블의 크기가 증가할수록 비분산 환경과 분산 환경의 속도 차이는 점점 커지는 것을 알 수 있다.

그림 13은 k 값의 변화에 따른 *k*-익명화를 수행 시간을 비교한 실험 결과이다. 이 실험에서 원본 테이블의 크기는 0.5GB로 고정하였고, k 값은 100, 500, 1000을 변화를 주었다. 실험 결과 그래프에서 알 수 있듯이, k 값이 증가할수록 익명화 수행 시간은 증가함을 알 수 있다. 이는 k 값이 증가할수록, 준식별자 속성 값에 대한 일반화가 더 많이 진행되기 때문이다. 또한, 실험 결과를 통하여 k 값이 증가할수록, 비분산 환경과 분산 환경의 속도 차이는 점점 커지는 것을 알 수 있다. 그림 12와 15의 실험 결과는 본 논문에서 제안하는 스파크 분산 환경 기반 *k*-익명화 기법이 효율적으로 익명화를 수행할 수 있다는 것을 실험적으로 증명한다.

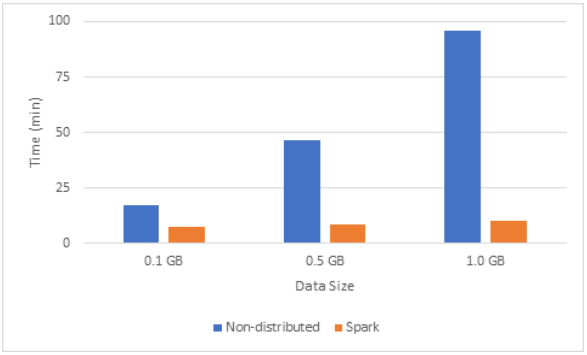

Fig. 12. Execution time comparison between non-distributed *k*-anonymity and distributed *k*-anonymity for varying number of records (k = 500)

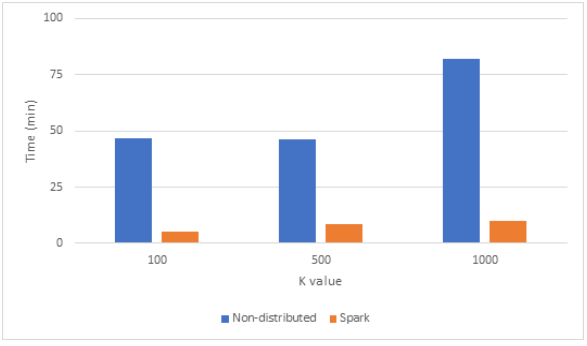

Fig. 13. Execution time comparison between non-distributed *k*-anonymity and distributed *k*-anonymity for varying number of k value (data size = 0.5 GB)

그림 14는 대용량 데이터에 대하여 본 논문에서 제안하는 스파크 분산 환경 기반 *k*-익명화 기법을 수행한 실험 결과이다. 이 실험에서 사용한 원본 테이블의 크기는 각각 2.5GB, 5GB, 100GB, 20GB에 해당한다. 또한, 실험에 사용한 k 값은 100, 500, 1000에 해당한다.

그림 14의 실험 결과는 다음과 같이 정리할 수 있다. 대체적으로 원본 테이블의 크기가 커질수록 익명화 소요 시간이 증가하였다. 하지만, k값이 100일 때 원본 테이블의 크기가 10GB, 20GB인 부분에서 익명화 수행 시간이 줄어드는 현상을 보였다. 이는 100으로 설정된 k값이 원본 테이블내의 레코드 수에 비해 너무 낮게 설정되어있기 때문이다. 데이터 크기가 커지면 격자에 맞게 일반화를 했을 때 같은 일반화된 레코드가 나올 확률이 커진다. 때문에 k가 100일 때 데이터 크기가 커짐에도 불구하고 속도가 줄어든다.

그림 14에서 데이터 크기가 20GB 일 때의 소요 시간과 그림 12의 비분산 환경 환경에서 실행한 익명화 소요시간을 비교하면 데이터 크기가 20배 차이 지만 스파크 분산 환경에서 실행한 익명화가 더 빠르다. 이는 빅데이터 시대인 만큼 데이터가 점점 커진다면 비분산 환경에서 수행하는 *k*-익명화는 한계점이 있음을 실험에서 증명하였다.

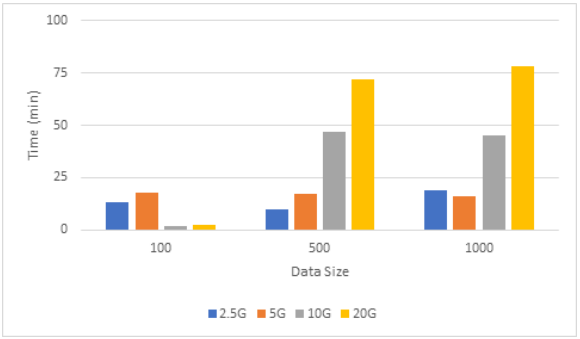

Fig. 14. *K*-Anonymity on Spark Distributed System

V. Conclusion

본 논문에서는 스파크 분산 환경에서 *k*-익명화 기법을 제시하였다. 본 논문에서 제시한 기법은 HDFS를 이용해서 원본데이터를 분산 저장하고, 얀과 스파크를 이용해 익명화 하는 과정을 분산 처리한다. 또한, 실 데이터를 이용한 실험을 통하여, 본 논문에서 제안한 기법이 기존의 비분산 익명화 기법보다 우수함을 알 수 있었다. 특히, 비분산 환경과 스파크 분산 환경간의 비교 실험에서 데이터 크기가 20배 차이가 나지만 소요 시간은 스파크 분산 시스템이 비분산 환경의 소요 시간보다 우수함을 알 수 있었다. 이는 빅데이터 익명화에 있어서 분산 시스템을 이용하는 경우, 비분산 환경을 통한 익명화보다 메모리를 효율적으로 사용하고 소요 시간도 더욱 줄어든다는 것을 암시한다.

REFERENCES

[1] A. Narayanan and V. Shmatikov, “Robust De-anonymi zation of Large Sparse Datasets”, In Proceedings of the 2008 IEEE Symposium on Security and Privacy Page, 2008.

[2] J. Kim, K.Jung, H. Lee, S. Kim, J.W. Kim and Y.D. Chung, “Models for Privacy-preserving Data Publishing : A Survey”, Journal of KIISE, Vol. 44, No. 2, pp. 195-207, 2017.

[3] B.C.M. Fung, K. Wang, R. Chen, and P.S. Yu, “Privacy-pres erving data publishing: A survey of recent developments”, ACM Computing Surveys, 42(4), June 2010.

[4] N. Mohammed, B.C.M. Fung, P.C.K. Hung, and C.K. Lee, “Centralized and distributed anonymization for high-dimensional healthcare data”, ACM Transactions on Knowledge Discovery from Data, 4(4), October 2010.

[5] L. Sweeney, “k-anonymity: A model for protecting privacy”, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, Vol. 10, No. 05, pp. 557-570 (2002)

[6] L. Sweeney, “Achieving k-Anonymity Privacy Protection using Generalization and Suppression”, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, Vol. 10, No. 05, pp. 571-588 (2002)

[7] K. LeFevre, D.J. DeWitt and R. Ramakrishnan, “Incognito: Efficient full domain k-anonymity”, In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2005.

[8] J. Byun, A. Kamra, E. Bertino, N. Li, “Efficient k-Anonym ization Using Clustering Technique”, DASFAA 2007: Advances in Databases: Concepts, Systems and Applications pp 188-200, 2007

[9] S. Kim, H. Lee, Y.D. Chung, “Privacy-preserving data cub for electronic medical records: An experimental evaluation”, International Journal of medical Informatics, 2017

[10] Apache Hadoop 2.8.4 API docs[Website]. *https://hadoo p.apache.org/docs/r2.8.4/* (2018)

[11] Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz, “Efficient big data processing in Hadoop MapReduce”, Proceedings of the VLDB Endowment, Volume 5 Issue 12, August 2012, Pages 2014-2015

[12] Apache Spark 2.3.0 API docs[Website]. *https://spark.ap ache.org/docs/2.3.0/index.html* (2018)

[13] Health Insurance Review and Assessment Service in Korea. http://opendata.hira.or.kr (2012).

Authors

Tae-Su Kim is a B.S. and M.S. student in the Department of Computer Science, Sangmyung University, Seoul, Korea, as of 2018. His research mainly focuses on data privacy, big data processing and distributed database system.

Jong Wook Kim is an assistant professor of Computer Science at Sangmyung University. His research interests include Web data mining, information retrieval, and database systems. Kim has a PhD from the School of Computing, Informatics, and Decision Systems Engineering at Arizona State University. He was a software engineer in Teradata Corporation. He is a member of the IEEE and the ACM.