하둡에서 데이터 접근 제어 설계 및 구현

김희주, 손시운, 길명선, 문양세
강원대학교 컴퓨터과학과
e-mail : {heeju, ssw5176, gils, ysmoon}@kangwon.ac.kr

Design and Implementation of Data Access Control in Hadoop

Heeju Kim, Siwoon Son, Myeong-Seon Gil, and Yang-Sae Moon
Dept. of Computer Science, Kangwon National University

요 약

최근 이슈가 되고 있는 하둡(hadoop) 패키지에 접목하여 많은 프로젝트들이 생겨나고 있으며, 이들 중 주요하게 떠오르고 있는 분야가 접근 제어 기술이다. 특히, 인터넷의 발전과 스마트 기기 사용자가 늘어남에 따라 데이터의 양이 증가하여, 데이터의 소유자와 사용자의 필요에 의한 접근 제어 기술이 필요하게 되었다. 본 논문에서는 접근 제어 기술의 필요성을 기반으로 HDFS(Hadoop Distributed File System, 하둡 분산 파일 시스템) 기반의 새로운 데이터 접근 제어 프레임워크를 제안한다. 제안하는 방법은 새로운 메타데이터 저장 모듈과 접근 관리 모듈을 만들어 데이터 접근 제어 프레임워크를 구성함으로써, 빅데이터 플랫폼을 사용하는 사용자들을 위한 접근 제어 기능을 제공한다. 제안한 프레임워크는 기존 플랫폼에 추가적인 설치가 필요 없도록 하둡 내부에 설계하여 향후 활용도가 높을 것이라 기대된다.

1. 서론

최근 인터넷이 발전하고 관련된 스마트 기기들이 다양하게 쏟아져 나오면서, 이를 사용하는 많은 사람들을 통해 개인 데이터의 양이 천문학적으로 증가하고 있다. 또한, 여러 분야에서 발생하는 다양한 대용량 데이터들로 인해 이른바 빅데이터(BigData)의 시대를 맞이하게 되었다. 이와 동시에 빅데이터 처리 기술 중 가장 대표적인 하둡(Hadoop) 기술도 함께 주목 받고 있다[1].

초기의 하둡 버전에서는 사용자가 파일을 저장하는 시스템인 HDFS(Hadoop Distributed File System, 하둡 분산 파일 시스템)를 안전한 환경에서 사용하는 것을 전제로 두었고, 이를 위해 데이터 유실 방지를 위한 접근 제어가 사용되었다[2, 3]. 그러나 하둡 플랫폼 사용자가 늘어나면서, 데이터 소유자들은 특정 사용자들의 접근을 원하지 않고, 데이터 사용자들은 특정 데이터의 접근만을 원하는 등의 요구가 증가하고 있다. 이로 인해 데이터의 소유자 및 사용자가 특정 권한을 기반으로 데이터를 사용하도록 하는 접근 제어 기술의 필요성이 더욱 높아지게 되었다.

본 논문에서는 기존 하둡 플랫폼에서 접근 제어 기술의 한계를 발견하고, 그에 따른 문제점을 해결하기 위해 하둡 기반의 새로운 데이터 접근 제어 프레임워크를 제안한다. 제안하는 프레임워크는 (1) 파일/디렉터리에 대한 사용자 권한 정보를 관리하는 새로운 메타데이터 저장 모듈, (2) 파일 접근 권한의 생성/수정/삭제 및 사용자 권한을 확인하는 접근 관리 모듈로 이루어져 있다. 이 두 가지 모듈을 이용하여 최종적으로 전체 접근 제어 프레임워크를 제안한다. 제안하는 프레임워크는 빅데이터 플랫폼을 사용하는 데이터 소유자와 사용자들을 위해 접근 제어를 제공하며, 기존 플랫폼을 그대로 사용 가능 하도록 설계하여 향후 활용도를 높이는 것을 목표로 한다.

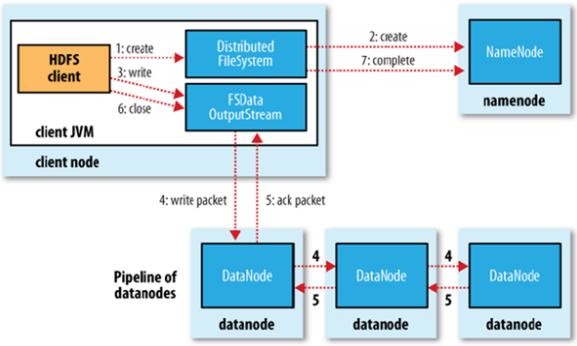
2. 관련연구

HDFS: HDFS 는 하둡 기반의 클라우드 컴퓨팅 분산처리에서 파일 시스템 구조를 담당하는 주요 구성 요소 중 하나이다. HDFS 의 기본 구조는 [그림 1]과 같다. HDFS 는 효과적인 분산 저장을 위해 마스터 기능을 하는 네임노드와 슬레이브 기능을 하는 여러 대의 데이터노드를 운용한다. 네임노드는 파일 시스템의 네임 스페이스를 관리하여 파일시스템 트리와 그 트리 안에 있는 파일/디렉터리에 대한 메타데이터를 유지한다. 데이터노드는 실질적으로 네임노드의 요청이 있을 때 블록을 저장하고 탐색하는 역할을 한다[4, 5].

사용자 인증: 하둡에서는 별도로 사용자 계정을 만들거나 관리하지 않고 리눅스 사용자 계정이나 커버로스(Kerberos)를 이용하여 사용자 인증을 수행한다. 첫 번째, 리눅스 사용자 계정을 이용한 인증방법은 HDFS 내에서 하둡 데몬을 실행하는 리눅스 계정을 활용하는 방법이다. 리눅스 계정을 활용하는 방법은 리눅스 계정을 하둡 사용자로 인증하고, 하둡 내에서는 별도의 접근 관리 기능을 갖지 않는다. 따라서, 하

† 본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었습니다.

둡 사용자 별로 접근 관리를 실행하기 위해서는 리눅스 사용자 계정 별로 디렉터리를 생성하고, 읽기/쓰기/실행 체계로 작성되어있는 POSIX 접근 목록에 폴더의 접근 권한을 추가적으로 설정해야 한다. 두 번째, 커버로스를 이용한 인증방법은 시스템에 접근하려는 사용자 혹은 서비스가 허가된 사용자인지 확인하는 보안방법이다. 커버로스는 서비스를 제공하는 서버와 서비스를 제공 받는 사용자가 신뢰할 수 있는 키 분배 센터인 KDC(Key Distribution Center)를 통하여 서로를 인증하는 프로토콜이며[6], 하둡의 네임노드와 클라이언트가 최초로 인증할 때 동작하도록 되어있다.



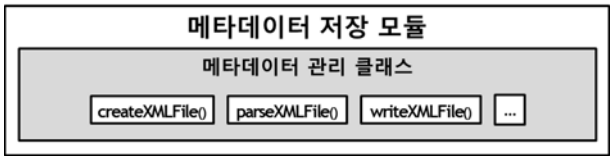
[그림 1] HDFS 에서 파일 저장과 복제[4].

3. HDFS 에서의 데이터 접근 제어 프레임워크

제안하는 데이터 접근 관리 프레임워크는 크게 사용자 권한 정보를 관리하는 메타데이터 저장 모듈, 파일과 사용자 접근 권한을 제어하는 접근 관리 모듈 두 가지로 구성된다.

3.1 메타데이터 저장 모듈 설계 및 구현

메타데이터 저장 모듈은 HDFS 접근 관리에 대한 메타데이터, 즉, 각 사용자의 파일/디렉터리에 대한 권한 정보를 저장한다. 특히, 권한 정보 메타데이터는 기존 읽기/쓰기/실행 체계로 작성되어있는 POSIX 접근 목록 기반 제어가 아닌 새로운 권한 체계에 대한 규칙을 만들어, 해당 규칙이 잘 반영될 수 있는 구조를 가진다. 모듈의 구조는 [그림 2]와 같이 HDFS 접근 관리 메타데이터의 구조를 정의한 XML 파일을 관리하는 클래스로 구성된다.



[그림 2] 메타데이터 저장 모듈 구조.

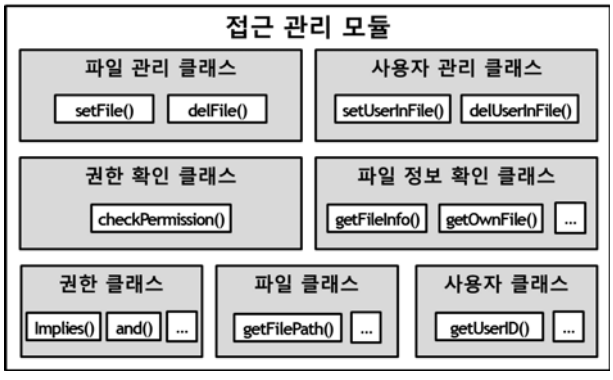
[표 1]은 정의된 접근 관리 메타데이터의 설계 구조를 설명한다. 메타데이터의 설계 구조는 파일/디렉터리의 소유자와 일반 사용자의 권한을 관리하기 위하여 소유자의 권한은 파일을 생성하는 사용자의 권한이기 때문에 일반 사용자의 권한보다 낮을 수 없으며 일반 사용자의 권한 역시 소유자가 가지고 있는 권한보다 높을 수 없다는 제약을 가진다.

[표 1] 메타데이터 설계 구조.

메타데이터	속성	정의 및 예시	
소유자 체크 테이블	fpath	정의	파일 혹은 디렉터리의 path *fpath는 파일 혹은 디렉터리가 될 수 있음
		예	fpath="/../../a.txt"
	type	정의	fpath의 type(파일 혹은 디렉터리)
		예	type="f" or type="d"
	uid	정의	User 중, 파일 소유자의 id
		예	uid="moon"
	opermission	정의	파일 소유자의 permission *opermission은 upermission의 권한보다 하위 권한일 수 없음
		예	opermission="rwx"
사용자 권한 체크 테이블	fpath	정의	파일 혹은 디렉토리의 path *fpath는 파일 혹은 디렉터리가 될 수 있음
		예	fpath="/../../a.txt"
	uid	정의	User의 id
		예	uid="moon"
	upermission	정의	User의 permission *upermission은 opermission의 권한보다 상위 권한일 수 없음
		예	upermission="rwx"

3.2 접근 관리 모듈 설계 및 구현

제안한 접근 관리 모듈은 향후 활용도를 고려하여 기존 플랫폼 내부를 수정하는 방법으로 설계하였다. [그림 3]은 이러한 접근 관리 모듈의 구조를 나타낸다. 접근 관리 모듈은 크게 파일 관리 클래스, 사용자 관리 클래스, 권한 확인 클래스, 파일 정보 확인 클래스, 권한 클래스, 파일 및 사용자 클래스로 구분된다.



[그림 3] 접근 관리 모듈 구조.

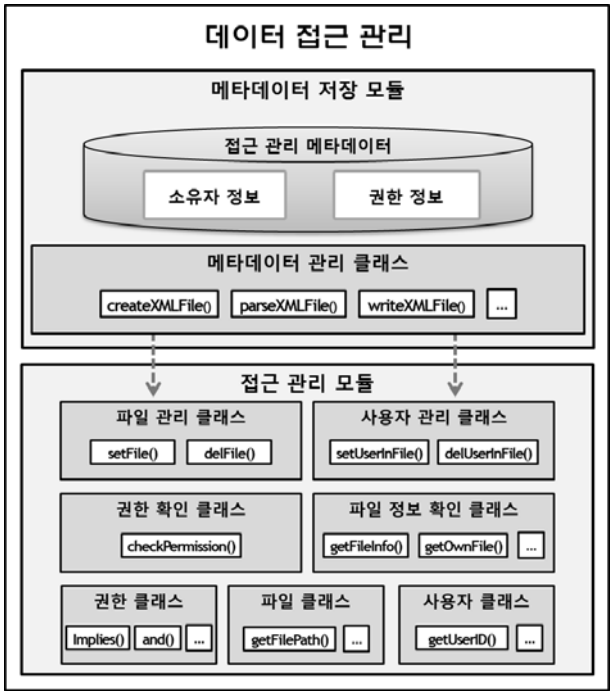
먼저, 파일 관리 클래스는 접근 제어를 수행하기 위한 HDFS 내의 파일/디렉터리의 주소, 소유자 ID,

소유자 권한을 설정하는 setFile() 함수와 파일/디렉터리를 제거하는 delFile() 함수로 이루어져 있다. 본 클래스는 사용자가 접근을 요청한 HDFS 내의 파일 주소를 가져오고, 파일이 신규이거나 해당 파일에 대한 정보가 메타데이터에 없는 경우 메타데이터 저장 모듈에 관련 정보를 전달하는 역할을 한다. 두 번째로, 사용자 관리 클래스는 파일/디렉터리에 사용자 권한을 설정하는 setUserInFile() 함수와 사용자 권한을 제거하는 delUserInFile() 함수로 이루어져 있다. 이 클래스는 파일의 소유자가 자신의 파일에 접근 권한을 승인할 사용자의 정보를 해당 파일의 소유자 권한과 비교하여 생성/변경/삭제하는 기능을 수행한다. 또한 사용자 관리 클래스는 앞서 언급한 파일 관리 클래스와 함께 메타데이터 관리 클래스의 동작 기반이 되는 정보들을 관리한다. 세 번째로, 권한 확인 클래스는 파일 접근을 요구하는 사용자의 권한을 확인하는 checkPermission()함수로 이루어져 있다. 이 클래스는 파일에 접근을 요청한 사용자 ID 를 하둡 사용자로부터 전달 받고, 접근 관리 메타데이터에서 해당 사용자의 권한 여부를 확인할 수 있도록 한다. 네 번째로, 파일 정보 확인 클래스는 XML 에 저장되어 있는 파일의 정보를 파일 및 사용자 클래스의 객체 및 객체 리스트로 반환하는 역할을 한다. 다섯 번째로, 권한 클래스는 접근 권한을 객체로 표현하거나 다른 객체와 논리 연산을 수행한다. 또한 기본적으로 사용자에게 부여할 수 있는 권한 메커니즘을 정의하는 역할을 하며 메타데이터에 정의된 권한에 대해 사용자가 수행할 작업의 적정성을 검토하는 기능도 수행한다. 마지막으로, 파일 및 사용자 클래스는 파일 정보 확인 클래스가 반환하는 정보를 표현하기 위한 매개체 역할을 하는 일종의 보조 클래스라 할 수 있다. 각 클래스가 생성하는 객체는 파일 및 사용자의 정보를 저장하며, 파일 객체는 사용자 객체를 리스트 형태로 포함할 수 있다.

3.3 전체 접근 제어 프레임워크 설계 및 구현

메타데이터 저장 구조 모듈과 접근 관리 모듈을 사용한 전체 데이터 접근 제어 프레임워크는 [그림 4]와 같다.

효과적인 접근 관리를 위하여 데이터 접근 관리의 기능을 네 가지로 나누어 수행한다. 첫 번째는 파일/디렉터리에 대한 권한 요청 시, 데이터 소유자의 허가 하에 특정 사용자의 권한을 생성해 주는 기능이다. 두 번째는 파일/디렉터리에 대한 권한 수정 요청 시, 데이터 소유자의 허가 하에 특정 사용자의 권한을 수정해 주는 기능이다. 세 번째는 파일/디렉터리에 대한 권한 삭제 요청 시, 특정 사용자의 권한을 삭제해 주는 기능이다. 마지막으로 네 번째는 파일/디렉터리의 권한 정보를 저장된 메타데이터를 통해 체크하는 기능이다. 모든 기능들은 데이터 소유자/관리자에 의해 수행 되며, 이 기능들의 동작 흐름은 각각 [그림 5], [그림 6], [그림 7], [그림 8] 과 같다(아래에서 “파일”은 파일/디렉터리를 포함).

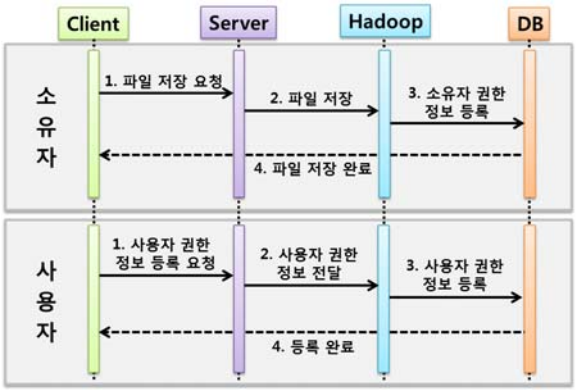


[그림 4] 접근 제어 프레임워크 구조.

첫째, 파일에 대한 권한의 생성 방법은 다음과 같다.

① 파일이 생성되는 경우 소유자, 파일 정보를 전달받아 파일 관리 클래스를 통해 접근 관리 메타데이터에 저장한다.

② 사용자가 접근 권한을 요청하는 경우, 소유자의 승인 하에 사용자 정보, 요청 권한, 파일 정보를 전달받아 사용자 관리 클래스 및 메타데이터 접근 모듈을 통해 해당 사용자의 권한을 생성한다.



[그림 5] 파일에 대한 권한의 생성 기능 흐름도.

둘째, 파일에 대한 권한의 수정 방법은 다음과 같다.

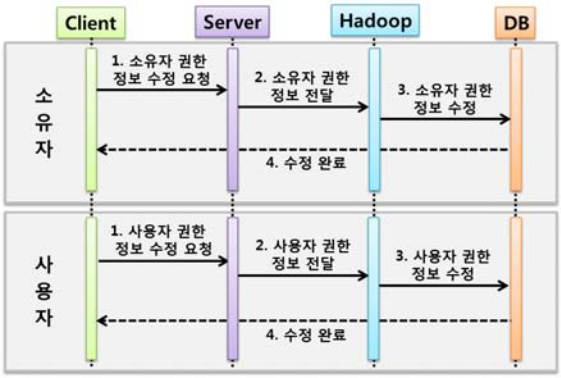
① 소유자가 접근 권한 수정을 요청하는 경우, 요청 권한, 접근 요청 파일 정보를 전달 받아 소유자의 권한을 수정한다.

② 또한, 수정된 소유자 권한에 따라 사용자에 대한 접근 권한을 승격/강등 시킨다.

③ 사용자가 접근 권한 수정을 요청하는 경우, 소유자의 승인 하에 사용자 정보, 요청 권한, 파일 정보를 전달 받아 사용자의 권한을 수정한다.

④ 소유자가 파일의 소유권을 타 사용자에게 양도할 경우, 소유자는 해당 파일 정보와 자신의 ID 및 양도할 사용자의 ID 를 전달 받아 해당 파일의 소유권을 수정한다.

⑤ 소유자 또는 수정 권한이 있는 사용자가 파일 경로의 변경을 요청하는 경우, 파일 경로와 변경할 파일 경로 및 자신의 ID 를 전달 받아 해당 파일의 경로를 수정한다.



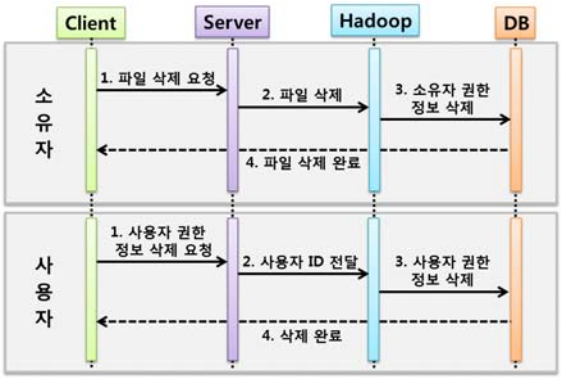
[그림 6] 파일에 대한 권한 수정 기능 흐름도.

셋째, 파일에 대한 권한의 삭제 방법은 다음과 같다.

① 소유자 또는 해당 파일의 삭제 권한이 있는 사용자가 파일 삭제를 요청하는 경우, 파일 정보를 전달 받아 접근 관리 메타데이터에서 삭제한다.

② 소유자가 특정 사용자의 권한을 회수할 경우, 해당 정보를 전달 받아 접근 관리 메타데이터에서도 이를 삭제한다.

③ 또한, 소유자 혹은 사용자가 삭제 되는 경우, 해당 사용자의 권한 정보 역시 메타데이터에서 삭제한다.

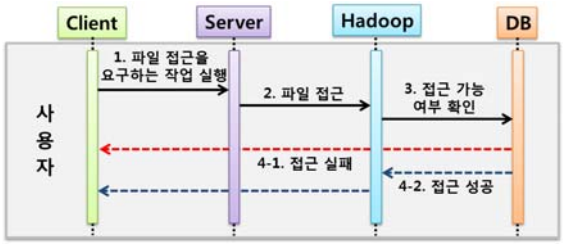


[그림 7] 파일에 대한 권한 삭제 기능 흐름도.

넷째, 파일에 대한 권한의 확인 방법은 다음과 같다.

① 특정 파일에 대한 접근(읽기/쓰기/실행)이 발생하는 경우, 접근하려는 사용자 및 파일 정보를 가져와 권한 확인 클래스를 통해 권한을 확인한다.

② 만약, 기존의 권한 정보에 의해 접근이 가능하다면, 권한 확인 클래스에서 사용자가 요청한 동작이 수행되도록 허가하고, 권한이 없다면 요청 동작을 수행하지 않고 클라이언트에게 실패 내용을 알린다.



[그림 8] 파일에 대한 권한 확인 기능 흐름도.

4. 결론 및 향후 연구

기존 빅데이터 플랫폼에서 접근 관리 기술의 한계에 기반하여, 본 논문에서는 HDFS 에서 사용자의 파일/디렉터리에 대한 권한 정보를 관리하는 새로운 메타데이터 저장 모듈을 제안하였다. 해당 모듈은 기존의 POSIX 접근 목록 기반 제어가 아닌 새로운 권한 체계에 대한 규칙을 만들어, 해당 규칙이 잘 반영될 수 있는 구조로 설계하였다. 또한, 파일 접근 권한에 대하여 소유자와 사용자 별로 접근 제어를 관리하는 접근 관리 모듈을 설계하고, 그 기능을 정의하였다. 위의 두 모듈로 이루어져있는 전체 프레임워크는 기존 빅데이터 플랫폼에서는 불가능했던 데이터 소유자 및 사용자별 접근 관리를 수행할 수 있을 뿐만 아니라, 기존 플랫폼을 그대로 사용 가능 하도록 설계하여 향후 활용도를 높인 매우 우수한 연구라 사료된다.

향후 연구로는 현재 시스템에서 XML 형태로 저장되어 있는 메타데이터를 Maria DB, HBASE 등으로 데이터베이스화할 예정이다. 또한, 권한 세분화, 사용자 계정 관리 등을 추가로 연구하여 다양한 활용이 가능한 프레임워크를 설계할 예정이다. 이를 통해 메타데이터의 데이터 무결성 및 동시성 제어를 확보하고, 조직 구조와 접근제어 요구사항을 반영할 수 있게 될 것이다.

참고문헌

[1] Apache Hadoop, http://hadoop.apache.org.

[2] K. Shvachko, H. Huang, S. Radia, and R. Chansler, “The Hadoop Distributed File System,” In *Proc. of the 2010 IEEE 26th Symposium on Massive Storage Systems and Technologies*(*MSST*), pp. 1-10, May 2010.

[3] D. Borthakur, “HDFS Architecture Guide,” HADOOP APACHE PROJECT, http://hadoop. apache. org/common/docs/current/hdfs_design.pdf, 2008.

[4] White, Tom, “Hadoop: The Definitive Guide,” O’Reilly Media, Inc., 2012.

[5] O. O’Malley, K. Zhang, S. Radia, R.Marti and C. Harrell, “Hadoop Security Design,” Yahoo, Inc., 2009.

[6] J. Kohi and C.Neuman, “The Kerberos Network Authentication Service(V5),” RFC 1510, Sept. 1993.