한국정보전자통신기술학회논문지(jkiiect)'21-2, Vol.14 No.1
논문 21-14-01-29 http://dx.doi.org/10.17661/jkiiect.2021.14.1.29

복잡한 구조의 데이터 중복제거를 위한 효율적인 알고리즘 연구

이협건*, 김영운, 김기영

Study of Efficient Algorithm for Deduplication of Complex Structure

Hyeopgeon Lee*, Young-Woon Kim, Ki-Young Kim

요 약 IT기술의 발달로 인해 발생되는 데이터양은 기하급수적으로 급격하게 증가하고 있으며, 데이터 구조의 복잡성은 높아지고 있다. 빅데이터 분석가와 빅데이터 엔지니어들은 이러한 빅데이터들을 보다 빠르게 데이터 처리 및 데이터 분석을 수행을 목표로 분석 대상의 데이터양을 최소화하기 위한 연구가 기업 및 가관 등 활발하게 이뤄지고 있다. 빅데이터 플랫폼으로 많이 활용되는 하둡은 서브프로젝트인 Hive를 통해 분석 대상의 데이터 최소화 등 다양한 데이터 처리 및 데이터 분석 기능을 제공하고 있다. 그러나 Hive는 데이터의 복잡성을 고려하지 않고 구현되어 중복 제거에 방대한 양의 메모리를 사용한다. 이에 복잡한 구조의 데이터 중복제거를 위한 효율적인 알고리즘을 제안한다. 성능평가 결과, 제안하는 알고리즘은 Hive에 비해 메모리 사용량은 최대 79%, 데이터 중복제거 시간은 0.677% 감소한다. 향후, 제안하는 알고리즘의 현실적인 검증을 위해 다수의 데이터 노드 기반 성능 평가가 필요하다.

Abstract The amount of data generated has been growing exponentially, and the complexity of data has been increasing owing to the advancement of information technology (IT). Big data analysts and engineers have therefore been actively conducting research to minimize the analysis targets for faster processing and analysis of big data. Hadoop, which is widely used as a big data platform, provides various processing and analysis functions, including minimization of analysis targets through Hive, which is a subproject of Hadoop. However, Hive uses a vast amount of memory for data deduplication because it is implemented without considering the complexity of data. Therefore, an efficient algorithm has been proposed for data deduplication of complex structures. The performance evaluation results demonstrated that the proposed algorithm reduces the memory usage and data deduplication time by approximately 79% and 0.677%, respectively, compared to Hive. In the future, performance evaluation based on a large number of data nodes is required for a realistic verification of the proposed algorithm.

Key Words : Big Data, Deduplication, Hadoop, MapReduce, Hive

1. 서론

IT기술의 발달로 인해 발생되는 데이터는 데이터의 양이 기하급수적으로 증가하고, 복잡성은 높아지고 있다. 빅데이터분석가와 빅데이터 엔지니어들은 이러한 빅데이터들을 보다 빠른 처리 및 분석을 위해 분석 대상의 최소화하기 위한 연구가 활발하게 이뤄지고 있다. 분석 대상의 최소화는 데이터분석가 및 빅데이터 엔지니어들이 사용하는 빅데이터 처리 및 분석 알고리즘의 처리 속도와 밀접한 관계가 있기 때문이다. 이러한 요구사항에 맞춰 Hive는 방대한 양의 데이터 분석 및 처리를 위한 하둡의 서브프로젝트 개발되어 활용된다[1][2].

그러나 Hive[3]는 데이터의 복잡성을 고려하지 않고, 관계형 데이터베이스에 적용된 일반적인 데이터 중복제거 알고리즘을 선택하여 제공되고 있다.

이로 인해 Hive 기반 데이터 중복제거 알고리즘

*Dept. of Data Analysis, Seoul Ganseo Campus of Korea Polytechnics
Dept. of Software Engineering, Seoil University
**Corresponding Author : Dept. of Data Analysis, Seoul Ganseo Campus of Korea Polytechnics (hglee67@kopo.ac.kr)
Received January 26, 2021 Revised January 29, 2021 Accepted February 07, 2021

[4][5]은 중복제거를 위한 데이터를 모두 메모리에 로드하여 각 데이터마다 중복 확인을 수행하며 처리한다. 또한 Hive 기반 데이터 중복제거 알고리즘[6][7]은 데이터 복잡성이 높아질수록 데이터의 모든 구조를 비교하며 연산한다. 따라서 Hive 기반 데이터 중복제거 알고리즘은 많은 양의 메모리를 요구하고 데이터 복잡성에 따라 처리시간이 증가된다.

이에 본 논문에서는 복잡한 구조의 데이터 중복제거를 위한 효율적인 알고리즘을 제안한다. 제안하는 알고리즘은 하둡 핵심 코어인 맵리듀스를 사용하여 데이터 중복제거 알고리즘을 구현한다. 제안하는 알고리즘의 크게 데이터 해시 단계와 데이터 중복제거 단계로 구성된다. 데이터 해시 단계는 맵리듀스의 맵 함수에서 데이터 중복제거를 위해 각 데이터들의 해시 알고리즘을 수행하여 데이터의 유일성을 확보한다. 데이터 중복제거 단계는 Shuffle and Sort를 통해 전달받은 결과를 기반으로 데이터의 중복을 제거한다.

성능평가 결과, 제안하는 알고리즘은 Hive에 비해 메모리 사용량은 최소3%에서 최대 79%까지 감소한다. 데이터 중복제거 처리시간은 제안하는 알고리즘은 Hive에 비해 데이터양이 600,000건 이하의 경우 약 2.857% 증가하지만, 700,000건부터 최대 0.677% 빠르게 처리한다.

본 논문의 구성은 다음과 같다. 2장에서는 Hive 기반 데이터 중복제거 알고리즘에 대해 살펴보고, 제안하는 알고리즘에 필요한 요구사항을 도출한다. 3장에서는 본 논문에서 제안하는 알고리즘을 구현하며, 4장에서는 제안한 알고리즘과 Hive와 성능을 분석한다. 마지막 5장에서는 결론 및 향후 연구 과제를 제시한다.

2. 관련 연구

본 장에서는 가장 보편적인 빅데이터 플랫폼인 하둡의 서브프로젝트인 Hive 기반 데이터 중복제거 알고리즘을 실험하고, 그 실험 결과를 기반으로 제안하는 알고리즘의 요구사항을 도출한다.

2.1 Hive 기반 데이터 중복제거 알고리즘

본 절에서는 보편적인 빅데이터 플랫폼인 하둡[8][9]의 서브 프로젝트 중 하나로 데이터 중복제거에 활용되는 Hive 기반 데이터 중복제거 알고리즘에 대해 살펴본다. Hive[10]는 하둡분산파일시스템에 저장되는 데이터를 기반으로 SQL과 유사한 HiveQL을 통해 데이터 저장, 처리, 분석을 수행한다. SQL을 이해하는 데이터분석가 및 데이터엔지니어는 Hive를 통해 쉽게 데이터 저장, 처리, 분석이 가능하다. Hive의 데이터 중복제거 알고리즘[2, 3]은 SQL과 동일하게 distinct 함수를 통해 수행한다. 그러나 distinct 함수의 수행 과정은 관계형 데이터베이스의 distinct 함수와 문법은 동일하지만, 수행 과정은 다르다. 그 이유는 Hive는 하둡분산파일시스템과 같은 분산 컴퓨팅 환경에서의 데이터 저장, 처리, 분석에 최적화하여 구현되었기 때문이다. 이로 인해 Hive 엔진은 하둡분산파일시스템의 다수의 데이터 노드에 저장된 모든 분석대상 데이터들을 메모리에 로드한 뒤 중복제거 알고리즘을 수행한다. 따라서 Hive 기반 데이터 중복제거 알고리즘은 많은 양의 메모리 사용을 요구한다.

〈표 1〉은 Hive 기반 데이터 중복제거 알고리즘의 요구사항 도출을 위한 실험환경을 나타낸다.

표 1. 실험 환경
Table 1. Experiment environment

Item	Setting
OS	Ubuntu 18
CPU	8 Core
RAM	32G
Swap Disk	20G
Main Disk	256 SSD
Hadoop Type	Stand Alone
Hive Metadata	MariaDB
Record Count	100,000 - 1,000,000
Column Count	100
Dimension	3 More
Duplication Rate	10%
Data Type	JSON

[그림 1]은 100,000건 데이터를 Hive 기반 중복제거 알고리즘을 실행한 예를 나타낸다.

```
hive> select distinct  * from test_tb;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be a
Query ID = myuser_20210121154211_dd4c5217-d122-496f-9f16-bf9
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input d
In order to change the average load for a reducer (in bytes)
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1610954377723_0013, Tracking URL = http:/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1
Hadoop job information for Stage-1: number of mappers: 1; nu
2021-01-21 15:42:15,807 Stage-1 map = 0%,  reduce = 0%
2021-01-21 15:42:20,923 Stage-1 map = 100%,  reduce = 0%, Cu
2021-01-21 15:42:26,046 Stage-1 map = 100%,  reduce = 100%,
MapReduce Total cumulative CPU time: 2 seconds 550 msec
Ended Job = job_1610954377723_0013
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 2.55 sec
Total MapReduce CPU Time Spent: 2 seconds 550 msec
OK
test_tb.col1    test_tb.col2    test_tb.col3
1       2       3
Time taken: 15.858 seconds, Fetched: 1 row(s)
hive>
```

그림 1. Hive를 통한 distinct 함수 실행의 예
Fig. 1. Example of distinct function using the hive

[그림 1]의 결과, Hive는 HiveQL의 구문을 컴파일하여 하둡의 데이터 처리 방법인 맵리듀스를 변환하여 데이터 중복제거를 수행한다. 맵리듀스의 맵 단계와 리듀스 단계에서 생성된 쓰레드들은 1개씩 생성되어 중복제거를 수행하며, 데이터 중복제거 처리시간은 15.858 sec이 발생한다. 맵의 쓰레드가 1개만 생성된 이유는 분석에 필요한 모든 데이터를 메모리를 로드해서 처리해야하기 때문이다.

Hive 기반 데이터 중복제거 알고리즘은 다음과 같다.

- 1단계. 하둡분산파일시스템으로부터 분석 대상으로부터 데이터 저장된 여러 파일읽기
- 2단계. 맵리듀스의 맵 함수를 통해 순서대로 파일의 데이터 레코드별 로드
- 3단계. 읽은 데이터를 메모리에 저장할 때, 기존에 저장된 데이터가 있는지 데이터 전체를 비교하여, 이미 저장된 데이터인지 확인하기
- 4단계. 중복되지 않은 데이터면 메모리에 저장
- 5단계. 메모리에 저장된 데이터를 리듀서로 보내 결과 파일 생성

2.2. 요구사항 도출

본 절은 앞서 선행 연구한 Hive의 실험 결과를 기반으로 요구사항을 도출한다.

2.2.1 메모리 사용량

실험에 사용한 레코드 수는 100,000건부터 1,000,000건까지 100,000건 단위로 레코드 수를 증가한다. 또한 레코드 수별 중복 레코드 비율은 10%로 정의하여 사용된 메모리양을 분석한다. 예를 들어 100,000건 레코드의 중복데이터는 10,000건이며, 중복 제거된 레코드 수는 90,000건이다.

〈표 2〉와 [그림 2]는 Hive의 레코드 수에 따른 사용된 메모리양을 나타낸다.

표 2. 레코드 수에 따른 사용된 메모리양
Table 2. Used memory by the record count

Record Count	Used Memory	Rate of Increase
100,000 row	487.33MB	-
200,000 row	1,017.33MB	209%
300,000 row	1,713.43MB	168%
400,000 row	2,299.91MB	134%
500,000 row	2,714.01MB	118%
600,000 row	3,204.87MB	118%
700,000 row	3,873.33MB	121%
800,000 row	4,397.33MB	114%
900,000 row	4,817.33MB	110%
1,000,000 row	5,230.91MB	109%

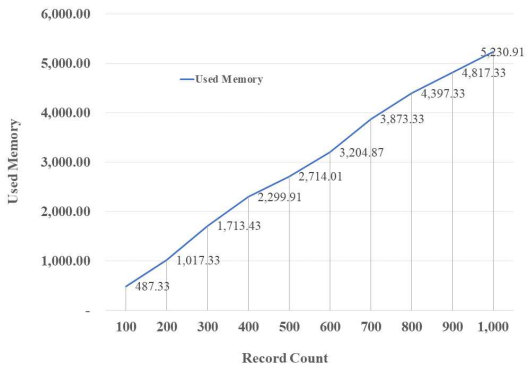

그림 2. 레코드 수에 따른 사용된 메모리양
Fig. 2. Used memory by the record count

〈표 2〉와 〈그림 2〉의 결과, Hive 기반 데이터 중복

제거 알고리즘은 데이터 중복제거를 수행하는 분석 대상의 데이터양이 증가할수록 사용된 메모리양이 급격하게 증가한다. 특히 1,000,000건 데이터 중복제거에 사용된 메모리양은 5,230.91MB로 100,000건 데이터 중복제거에 사용된 메모리양 대비 약 10.9배 증가한다. 또한 100,000건과 200,000건의 데이터 중복제거에 사용된 메모리양의 증가율은 209%로 약 2배 이상 증가한다. 사용된 메모리양의 증가율은 분석대상의 데이터가 증가할수록 감소한다. 그러나 증가율의 감소는 사용된 메모리양의 증가율 계산에 사용된 모수가 커지기 때문에 감소한 것이며, 사용되는 메모리양은 데이터양이 증가될수록 지속적으로 증가된다. 따라서 사용된 메모리양을 줄일 수 있는 데이터 중복 제거 방법은 필요하다.

2.2.2 복잡한 구조에 적합한 중복제거

Hive는 하둡분산파일시스템에 저장되는 데이터 중복제거를 위한 데이터 로딩하기 위해 키와 값으로 구성된 맵 구조로 데이터를 정의한다. 로딩되는 데이터의 키는 Offset 값을 저장하고, 값은 데이터 중복제거를 위한 데이터를 저장한다.

이로 인해 Hive 기반 데이터 중복제거 알고리즘은 값의 모든 데이터들을 메모리에 로딩하며 메모리에 존재하는 데이터인지 확인하는 방법으로 중복제거를 수행한다.

이러한 방식은 단순한 구조의 텍스트 데이터 중복제거는 크게 어렵지 않게 수행이 가능하다. 그러나 JSON, XML 등 복잡한 구조의 텍스트 데이터와 같은 다차원 데이터는 중복제거에 많은 리소스를 요구한다. 따라서 복잡한 구조의 데이터 중복제거 알고리즘이 필요하다.

3. 제안하는 데이터 중복제거 알고리즘

본 장에서는 앞서 도출된 요구사항을 기반으로 복잡한 구조의 데이터 중복제거를 위한 효율적인 알고리즘을 제안한다. 제안하는 알고리즘은 하둡의 핵심 코어인 맵리듀스를 활용하여 구현한다.

제안하는 알고리즘은 데이터 해시 처리 단계와 데이터 중복제거 단계로 구성된다. 데이터 해시 처리 단계는 맵리듀스의 맵 단계에서 수행하고, 데이터 중복제거는 맵리듀스의 리듀스 단계에서 수행한다. [그림 3]은 맵리듀스의 단계별 제안하는 데이터 중복제거 알고리즘의 단계를 나타낸다.

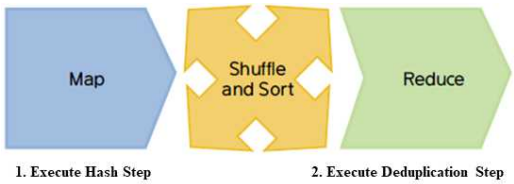

그림 3. 맵리듀스 단계별 제안하는 알고리즘 단계
Fig. 3. Proposal Algorithm Steps by Map-Reduce

3.1. 데이터 해시 단계

데이터 해시 단계는 단순한 구조의 텍스트 데이터 및 복잡한 구조의 데이터 데이터에 영향을 받지 않도록 해시 함수를 활용하여 데이터 처리한다. 해시 함수는 단방향 암호화 함수로 데이터 무결성에 활용되는 함수이다.

데이터 해시 단계의 수행은 맵리듀스 프레임워크의 맵 단계에서 수행하여, 수행 단계는 다음과 같다.

- 1단계. 하둡분산파일시스템으로부터 분석 대상으로부터 데이터 저장된 여러 파일읽기
- 2단계. 맵리듀스의 맵 함수를 통해 순서대로 파일의 데이터 레코드별 읽으며, 쓰레드 수 증가
- 3단계. 읽은 레코드별 데이터를 각각 해시 함수로 암호화
- 4단계. 암호화된 해시 값은 Shuffle and Sort의 키로 정의하고, Shuffle and Sort의 값으로 전달

[그림 4]는 데이터 해시 단계의 의사코드를 나타낸다.

```
1  // Used map function in Map-Reduce
2  void map(Key, Value, Context){
3
4    // Excute Hash Algorithm by record
5    ssKey = encHash(Value);
6
7    // Shuffle and Sort Key : EncHash
8    // Shuffle and Sort Value : PlanText
9    context.wirte(ssKey, Value);
10 }
```

그림 4. 데이터 해시 단계의 의사코드
Fig. 4. Pseudo Code of the data hash step

데이터 해시 단계의 사용한 해시 알고리즘은 SHA-256 알고리즘을 사용한다. SHA-256 알고리즘을 선택한 이유는 보안성 강화를 목적으로 사용하지 않고 처리속도를 가중치를 부여하여 결정하기 때문이다. 추가 설명으로 데이터 해시 단계의 해시 알고리즘은 단순한 구조의 텍스트 데이터 및 복잡한 구조의 데이터 데이터에 영향을 받지 않고 빠르게 데이터의 유일성만 보장하면 되기 때문이다.

데이터의 유일성이 보장된 해시 알고리즘 결과는 맵리듀스의 Shuffle and Sort의 키로 정의하고, 레코드 데이터는 Shuffle and Sort의 값으로 정의하고 전달한다.

전달된 데이터는 Shuffle and Sort에 기본 기능을 따라 데이터를 변환된다.

3.2. 데이터 중복제거 단계

데이터 중복제거 단계는 맵리듀스의 Shuffle and Sort를 수행한 결과를 기반으로 리듀스 함수에서 데이터 중복제거를 처리한다. Shuffle and Sort는 앞서 제안한 데이터 해시 단계에서 전달한 키의 데이터를 기반으로 정렬을 수행한다. 그 뒤, Shuffle and Sort는 동일한 키에 대한 데이터들을 배열로 저장한다.

[그림 5]는 Shuffle and Sort 결과 데이터의 예를 나타낸다.

Shuffle and Sort	
Key(Enc Hash)	Value(Plan Text)
abcdefg	My Test Data1
	My Test Data1
	My Test Data1
hijklnm	My Test Data2
	My Test Data2
opgrstu	My Test Data3
	My Test Data3
	My Test Data3

그림 5. Shuffle and Sort 결과 데이터의 예
Fig. 5. Example of Shuffle and Sort's result

[그림 5]와 같이 Shuffle and Sort로 결과 데이터는 해시 알고리즘 결과를 키, 레코드 데이터는 키에 따른 동일한 레코드 데이터 값들이 배열로 저장된다. [그림 6]은 데이터 중복제거 단계의 의사코드를 나타낸다.

```
1  // Used ruduce function in Map-Reduce
2  void reduce(Key, Value[], Context){
3
4    // First data of Value array
5    context.wirte(Value[0], Null);
10 }
```

그림 6. 데이터 중복제거 단계 단계의 의사코드
Fig. 6. Pseudo Code of the data deduplication step

데이터 중복 제거 방법은 Shuffle and Sort의 결과 데이터 중 배열 첫 번째 값 데이터만 가져오면 된다. 그 이유는 배열로 저장된 데이터의 내용은 모두 동일하기 때문이다. 데이터 중복 제거 결과 데이터는 Context.write() 함수를 통해 하둡분산파일시스템에 저장하며, 중복제거 결과 파일을 생성한다.

4. 성능평가

본 장에서는 제안하는 데이터 중복제거 알고리즘의 성능평가를 위해 메모리 사용량과 데이터 중복제거 처리시간을 분석한다.

4.1 메모리 사용량 분석

메모리 사용량 분석은 제안한 알고리즘을 구현하여 앞서 관련연구에서 정의한 실험 환경에 맞춰 성능 평가를 수행한다. 제안하는 알고리즘은 *P.A*로 정의하고 Hive와 성능평가한다. 〈표 3〉과 〈그림 6〉은 제안한 알고리즘의 레코드 수에 따른 메모리 사용량 분석 결과를 나타낸다.

표 3. 레코드 수에 따른 사용된 메모리양
Table 3. Used memory by the record count

Record Count	Used Memory		Rate
	P.A	Hive	
100,000 row	471.96MB	487.33MB	-3%
200,000 row	590.15MB	1,017.33MB	-42%
300,000 row	711.13MB	1,713.43MB	-58%
400,000 row	835.76MB	2,299.91MB	-64%
500,000 row	942.11MB	2,714.01MB	-65%
600,000 row	961.54MB	3,204.87MB	-70%
700,000 row	989.43MB	3,873.33MB	-74%
800,000 row	991.34MB	4,397.33MB	-77%
900,000 row	1,001.31MB	4,817.33MB	-79%
1,000,000 row	1,081.55MB	5,230.91MB	-79%

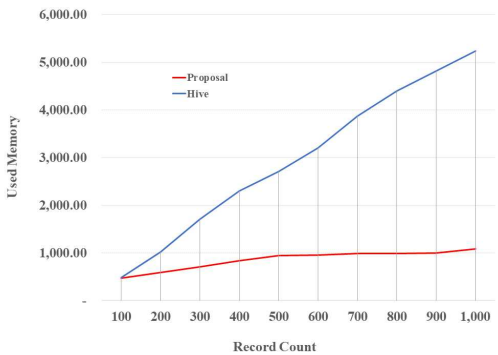

그림 6. 레코드 수에 따른 사용된 메모리양
Fig. 6. Used memory by the record count

메모리 사용량 분석 결과, 제안하는 데이터 중복 제거 알고리즘은 Hive보다 최소 3%에서 최대 79%의 메모리를 사용량이 감소한다. 특히 제안하는 데이터 중복제거 알고리즘은 데이터양이 증가할수록 메모리 사용량이 급격하게 감소한다.

이러한 결과가 나온 이유는 Hive 기반 데이터 중복제거 알고리즘은 데이터의 중복 확인을 위해 모든 데이터를 메모리에 저장하여 처리를 수행한다. 따라서 Hive 기반 데이터 중복제거 알고리즘은 데이터양이 증가할수록 메모리 사용량도 급격하게 증가한다.

4.2 데이터 중복제거 처리시간 분석

데이터 중복제거 처리시간 분석은 제안한 알고리즘을 구현하여 앞서 관련연구에서 정의한 실험 환경에 성능 평가를 수행한다. 중복제거 대상데이터는 JSON 형태로 구현된 3차원 이상의 데이터를 사용한다. 〈표 4〉와 〈그림 7〉은 제안한 알고리즘의 레코드 수에 따른 데이터 중복제거 처리속도 분석 결과를 나타낸다.

표 4. 레코드 수에 따른 데이터 중복제거 처리시간
Table 4. Processing time of deduplication by the record count

Record Count	Processing time		Rate
	P.A	Hive	
100,000 row	16.134ms	15.858ms	1.740%
200,000 row	22.253ms	21.123ms	5.347%
300,000 row	27.114ms	26.134ms	3.750%
400,000 row	34.134ms	33.132ms	3.024%
500,000 row	39.312ms	38.132ms	3.093%
600,000 row	41.399ms	41.321ms	0.189%
700,000 row	45.441ms	45.542ms	-0.222%
800,000 row	48.999ms	49.123ms	-0.253%
900,000 row	54.931ms	55.134ms	-0.368%
1,000,000 row	58.130ms	59.234ms	-1.864%

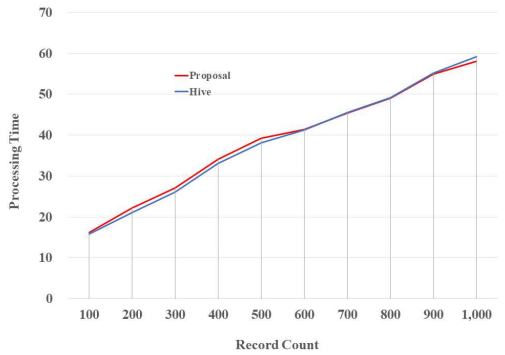

그림 7. 레코드 수에 따른 데이터 중복제거 처리시간
Fig. 7. Processing time of deduplication by the record count

데이터 중복제거 처리시간 분석 결과, Hive는 제안하는 데이터 중복 제거 알고리즘 보다 100,000건부터 600,000건까지 약 2.857% 빠르게 데이터를 중복 제거를 수행한다. 이러한 결과 발생한 이유는 데이터 중복제거에 필요한 데이터의 저장 위치가 Hive는 메모리에 저장되고, 제안하는 알고리즘은 일반 디스크에 저장되기 때문에 발생한다. 그러나 Hive의 메모리 사용량에 대비 약 2.857% 처리시간 차이는 제안하는 알고리즘에 비해 처리시간이 크지 않아 메모리 사용에 대한 이점이 크게 없다.

특히 데이터 중복제거 처리시간은 700,000건부터 1000,000건까지 데이터양이 증가될수록 제안하는 알고리즘이 Hive에 비해 약 0.677% 빠르게 데이터 중복제거를 수행한다. 이러한 이유가 발생한 이유가 데이터양에 증가에 따라 메모리에 저장된 모든 데이터에 대해 중복 확인과정의 횟수가 급격하게 증가되었기 때문이다. 또한 복잡한 구조의 데이터 구조의 중복제거는 데이터 차원별 구조를 비교하여 중복을 확인하기 때문에 처리시간이 증가한다.

5. 결론

본 논문에서는 본 논문에서는 복잡한 구조의 데이터 중복제거를 위한 효율적인 알고리즘을 제안한다. 제안하는 알고리즘은 하둡 핵심 코어인 맵리듀스를 사용하여 중복제거 알고리즘을 구현하며, 데이터 해시 단계와 데이터 중복제거 단계로 구성한다. 데이터 해시 단계는 맵리듀스의 맵 함수에서 데이터 중복제거를 위해 각 데이터들의 해시 알고리즘을 수행하여 데이터의 유일성을 확보한다. 데이터 중복제거 단계는 Shuffle and Sort를 통해 전달받은 결과를 기반으로 데이터의 중복을 제거한다.

성능평가 결과, 제안하는 알고리즘은 Hive에 비해 100,000건에서 최소3%, 1,000,000건에서 최대 79%까지 메모리 사용량이 감소한다. 또한 제안하는 알고리즘의 메모리 사용량은 데이터의 수가 증가될수록 Hive에 비해 급격하게 감소한다. 데이터 중복제거 처리시간은 Hive에 비해 데이터양이 600,000건 이하의 경우 약 2.857% 증가하였지만, 700,000건부터 최대 0.677% 빠르게 처리한다.

향후, 제안하는 알고리즘의 현실적인 검증을 위해 다수의 데이터 노드 기반 성능 평가가 필요하다.

REFERENCES

[1] H. G. Lee, Y. W. Kim, K, Y. Kim "Study of In-Memory based Hybrid Big Data Processing Scheme for Improve the Big Data Processing Rate", Journal of Korea Institute of Information, Electronics, and Communication Technology, 12(2), pp. 127-134, April, 2019
[2] In-Hak Joo, "Spatial Big Data Query Processing System Supporting SQL-based Query Language in Hadoop," Journal of Korea Institute of Information, Electronics, and Communication Technology, 10(1), pp.1-8, February, 2017
[3] H. G. Lee, Y. W. Kim, K. Y. Kim "Design of GlusterFS Based Big Data Distributed Processing System in Smart Factory", Journal of Korea Institute of Information, Electronics, and Communication Technology, 11(1), pp.70-75, February, 2018
[4] H. G. Lee, Y. W. Kim, K. Y. Kim, "Implementation of an Efficient Big Data Collect ion Platform for Smart Manufacturing," Journal of Engineering and Applied Sciences, 12(2Si), pp.6304-6307, 2018
[5] Yue Liu, Shuai Guo, Songlin Hu, Tilmann R

abl, Hans-Arno Jacobsen, Jintao Li, Jiye Wang, "Performance Evaluation and Optimization of Multi-Dimensional Indexes in Hive," IEEE Transactions on Services Computing, 11(5), pp.835-849, July, 2016

[6] Xi Peng, Liang Liu, Lei Zhang, "A Hive-Based Retrieval Optimization Scheme for Long-Term Storage of Massive Call Detail Records," IEEE Access, Vol.8, pp.431-444, December, 2019

[7] Mudassar Ahmad, Safina Kanwal, Maryam Cheema, Muhammad Asif Habib, "Performance Analysis of ECG Big Data using Apache Hive and Apache Pig," 2019 8th International Conference on Information and Communication Technologies(ICICT), November, 2019

[8] Jongyeop Kim, Seongsoo Kim, Donghoon Kim, Hong Liu, "Automated Configuration Parameter Classfication Model for Hive Query Plan on the Apache Yarn," 2019 IEEE International Conference on Big Data, Cloud Computing, Data Science & Engineering (BCD), May, 2019

[9] Zhiang Wu, Aibo Song, Jie Cao, Junzhou Luo, Lu Zhang, "Efficiently Translating Complex SQL Query to MapReduce Jobflow on Cloud," IEEE Transactions on Cloud Computing, 8(2), pp.508-517, May, 2017

[10] Fan Zhang, Majd F. Sakr, Kai Hwang, Samee U. Khan, "Empirical Discovery of Power-Law Distribution in MapReduce Scalability," IEEE Transactions on Cloud Computing, 7(3), pp.744-755, February, 2017

저자약력

이 협 건(Hyeopgeon Lee) **[종신회원]**

- 2011.03 - 2015.08, 숭실대학교 일반대학원 컴퓨터학과 공학박사
- 2015.12 - 현재, 한국폴리텍대학 서울강서캠퍼스 데이터분석과 교수

〈관심분야〉 빅데이터, 실시간 분석, 데이터분석

김 영 운(Young-Woon Kim) **[종신회원]**

- 2004.09 - 2018.08, 숭실대학교 일반대학원 컴퓨터학과 공학박사
- 2015.12 - 현재, 한국폴리텍대학 서울강서캠퍼스 데이터분석과 교수

〈관심분야〉 빅데이터, 실시간 분산 처리, 데이터분석

김 기 영(Ki-Young Kim) **[종신회원]**

- 2004.03 - 현재, 서일대학교 교수

〈관심분야〉 빅데이터, 무선통신, 사물인터넷