Journal of Convergence for Information Technology
Vol. 10. No. 10, pp. 1-8, 2020
e-ISSN 2586-4440
DOI : https://doi.org/10.22156/CS4SMB.2020.10.10.001

쿠쿠 필터 유사도를 적용한 다중 필터 분산 중복 제거 시스템 설계 및 구현

김영아[1], 김계희[2], 김현주[3], 김창근[3*]
[1]엔코아 데이터 HRD 본부 연구원 , [2]경남과학기술대학교 컴퓨터공학과 강사, [3]경남과학기술대학교 컴퓨터공학과 교수

Design and Implementation of Multiple Filter Distributed Deduplication System Applying Cuckoo Filter Similarity

Yeong-A Kim[1], Gea-Hee Kim[2], Hyun-Ju Kim[3], Chang-Geun Kim[3*]
[1]Researcher, Data HRD Headquarters , EN-CORE.Co.,Ltd.
[2]Lecturer, Department of Computer Science & Engineering, GNTECH
[3]Professor, Department of Computer Science & Engineering, GNTECH

요 약 최근 몇 년 동안 기업이 수행하는 비즈니스 활동에서 생성된 데이터를 기반으로 하는 기술이 비즈니스 성공의 열쇠로 부상함에 따라 대체 데이터에 대한 저장, 관리 및 검색 기술에 대한 필요성이 대두되었다. 기존 빅 데이터 플랫폼 시스템은 대체 데이터인 비정형 데이터를 처리하기 위해 실시간으로 생성된 대량의 데이터를 지체 없이 로드하고 중복 데이터 발생 시 서로 다른 스토리지의 중복 제거 시스템을 활용하여 스토리지 공간을 효율적으로 관리해야 한다. 본 논문에서는 빅 데이터의 특성을 고려하여 쿠쿠 해싱 필터 기법의 유사도를 이용한 다중 계층 분산 데이터 중복 제거 프로세스 시스템을 제안한다. 가상 머신 간의 유사성을 쿠쿠 해시로 적용함으로써 개별 스토리지 노드는 중복 제거 효율성으로 성능을 향상시키고 다중 레이어 쿠쿠 필터를 적용하여 처리 시간을 줄일 수 있다. 실험 결과 제안한 방법은 기존 블룸 필터를 이용한 중복 제거 기법에 의해 8.9%의 처리 시간 단축과 중복 제거율이 10.3% 높아짐을 확인하였다.

주제어 : 분산중복제거, 빅 데이터, 쿠쿠 해시, 다중계층 쿠쿠 필터, 소프트웨어 스토리지

Abstract The need for storage, management, and retrieval techniques for alternative data has emerged as technologies based on data generated from business activities conducted by enterprises have emerged as the key to business success in recent years. Existing big data platform systems must load a large amount of data generated in real time without delay to process unstructured data, which is an alternative data, and efficiently manage storage space by utilizing a deduplication system of different storages when redundant data occurs. In this paper, we propose a multi-layer distributed data deduplication process system using the similarity of the Cuckoo hashing filter technique considering the characteristics of big data. Similarity between virtual machines is applied as Cuckoo hash, individual storage nodes can improve performance with deduplication efficiency, and multi-layer Cuckoo filter is applied to reduce processing time. Experimental results show that the proposed method shortens the processing time by 8.9% and increases the deduplication rate by 10.3%.

Key Words : Distributed Deduplication, Big Data, Cuckoo Hash, Multilayer Cuckoo Filter, Software Storage

*This work was supported by Gyeongnam National University of Science and Technology Grant in 2020~2021

*Corresponding Author : Chang-Geun Kim(cgkim@gntech.ac.kr)

Received August 18, 2020
Revised October 5, 2020
Accepted October 20, 2020
Published October 28, 2020

1. 서론

AI로 대두되는 데이터 과학, 기계 학습, 자연어 처리와 빅 데이터에 기반 모델링 등의 기술이 발전하면서 빅 데이터 중 대체 데이터가 확산하고 있다.

대체 데이터는 웹 사이트의 상품정보인 웹 데이터, 사용자의 방문 시간, 사용자 접속 수, 사용자 활동 데이터인 웹 트래픽, 지역 경제를 측정하는 위성 영상 데이터 등을 검색 API를 통해 데이터베이스에 통합 처리된다. 대체 데이터는 비정형 데이터로 구글, 야후 등과 같은 검색 엔진을 통해 수집· 축적 ·분석함으로써 기업의 경영 전략과 상품 개발에 유용하게 활용된다. IDC는 2025년에 총 데이터양이 163 제타 바이트(ZB)에 이를 것으로 전망했고, 이 중 80%는 비정형 데이터로 구성될 것으로 예측했다[1].

웹의 광대한 통신 대역을 활용하고 빠른 시간에 포괄적인 크롤링 작업과 분석에 필요한 관련 요구 사항 기술의 개선 및 정보 관리시스템, 병렬 수집 시스템 등의 개발로 데이터 처리 기능이 빠른 하드웨어를 요구하는 워크로드 중 Swift[2], Ceph[3], LeoFS[4]과 같은 오브젝트 스토리지 시스템이 주목을 받고 있다.

오브젝트 스토리지 시스템은 다양한 중복 제거 기법 알고리즘을 적용한 클러스터를 SDS(Software Defined Storage) 환경[5]으로 구축한 시스템이다. 가상화된 물리 서버를 마운트 하여 사용자 다수가 할당된 가상머신을 통해 저장 공간을 사용한다. 각 가상머신은 데이터를 파일 단위와 블록 단위가 아닌 객체단위로 관리되어 저장하고 해시테이블을 통해 메타데이터를 참조한다. 따라서 동일한 데이터를 여러 곳에 분산한 후 기록하기 때문에 중복 데이터를 처리하여야 한다. 중복 데이터 제거 기술은 중복 데이터가 발생할 경우 바이너리 타입의 원본 데이터에 정보를 부여하고 발생한 데이터를 삭제 후 데이터의 크기를 압축하는 기술이다. 이때 발생하는 중복 위치 검색은 이전의 많은 연구를 통해 적용하여 중복 데이터를 줄일 수 있는데 사용되고 있으나 대용량의 메모리와 처리 시간문제로 분산 스토리지 시스템에서 중복 제거를 사용하는 시스템은 알려지지 않았다. Swift, Ceph, LeoFS 등은 오픈소스로 공개가 되고 있지만 LeoFS는 데이터 손실 보상 및 중복 제거는 구현되지 않았고, Swift와 Ceph는 데이터 손실 보상은 구현되어 있지만 중복 제거는 구현되어 있지 않다. 상용 객체 스토리지로는 Atoms[6]과 Himalaya[7]가 존재하지만, 이들도 중복 제거 시스템은 구현되어 있지 않다. 또한, 클라우드 서비스로는 Amazon S3[8], Google CloudStorage[9], Liquid[10]는 중복 제거를 구현하고 있지만, 가상 머신 이미지에 최적화되어 있으며, 범용 객체 스토리지로 사용하는 것은 어렵다.

본 논문은 이러한 단점을 해결하고 SDS 환경에서 중복 제거 성능을 향상하기 위하여 클라이언트 서비스 객체인 유사도 기반 클러스터링과 쿠쿠 필터[11] 알고리즘을 적용한 다중계층 분산 중복제거 프로세스 시스템을 제안한다. 기존의 중복제거 시스템 중 데이터 집합 저장공간 테스트를 일반적인 확률의 유사도를 적용한 다중계층 불룸필터 분산 중복제거[12] 시스템에 비해 삽입, 삭제, 검색을 지문과 버킷의 크기로 식별되는 쿠쿠 필터의 유사도를 기반으로 다중 계층 쿠쿠 필터 분산 중복제거 시스템을 사용하여 SDS 도메인에 대한 중복 제거 실행 시 저장공간, 인덱스 조회시간, 중복 제거율의 감소 효과를 갖는다.

2. 관련연구

2.1 데이터 중복 제거(De-duplication)

데이터 중복 제거는 데이터 집합 내에서 중복 블록을 식별하고 제거하는 것을 말한다. 데이터 중복제거 시스템은 백업의 대상이 되는 데이터를 식별하는 시스템으로 클라이언트와 데이터를 저장하는 스토리지의 중간에 위치한다. 데이터는 이미지 파일, 문서 파일, 압축 파일, 구조화된 데이터베이스 등을 포함하며 백업 대상 데이터는 LAN (Local Area Network) 또는 SAN (Storage Area network)로 연결된 스토리지의 보관 매체에 기록된다.

중복 제거 시스템의 일반적인 방법은 중복 제거할 데이터를 연속적인 데이터 블록인 청크 단위로 분할한다. 분할된 데이터는 청크 알고리즘을 이용하여 데이터의 청크를 생성하고 해싱 엔진의 알고리즘을 통해서 고유한 식별 정보로서 청크에 대한 지문 인덱스를 생성한다. 지문 인덱스는 청크 별로 중복 여부를 판단하는 키를 말하며 SHA-1, SHA-256, MD5 등의 해시 함수로 계산한다 [13-17]. 지문 캐시는 기존의 지문 인덱스와 생성된 값이 일치하면 레시피 저장소에 저장하고 일치하지 않으면 컨테이너 저장소에 청크를 저장하게 된다.

중복 제거는 데이터를 백업하는 실행 장소와 시간에 따라 크게 전처리 방식, 인라인, 포스트 프로세스 방식

으로 나뉜다. 전처리 방식은 서버 측에서 데이터의 중복을 탐지한 후 중복 제거된 데이터를 스토리지에 전송하는 방식이다. 데이터 백업 시 변경 블록만 전송하므로 전체 백업 시 백업 시간을 단축하는 효과가 있다. 그러나 백업 대상이 되는 서버에서 처리가 실행되기 때문에 CPU 나 메모리 등의 추가 자원이 필요하다.

인라인 방식은 서버에서 스토리지에 데이터를 전송하는 과정에서 중복을 감지하고 저장하기 전에 중복 데이터를 제거하는 방식이다. 데이터가 저장되는 시점에서 중복 제거 처리도 동시에 실행되는 소프트웨어 부분과 스토리지에서 처리되는 하드웨어 부분으로 나누어 데이터가 처리된다. 소프트웨어 중복 제거 처리는 미디어 서버의 백업 소프트웨어에서 실행되며 중복 제거에 필요한 부하는 미디어 서버에 집중시킬 수 있다. 데이터를 전송하는 동안 중복 제거 작업을 수행하기 때문에 병목현상으로 데이터의 전송 성능이 낮아진다. 포스트 프로세스 방식은 사후 처리 방식이라고도 하며 서버의 스토리지에 모든 데이터를 저장한 후 스토리지에서 중복 제거를 하는 방식이다. 백업 된 데이터가 먼저 스토리지에 저장된 후 중복 제거 프로세스가 실행된다. 백업 속도가 빠르지만, 중복 제거 전의 데이터를 저장할 공간이 스토리지에 필요하다.

2.2 쿠쿠 해싱 필터 (Cuckoo Hashing Filter)

쿠쿠 해싱 필터는 삽입된 데이터의 지문을 저장하는 해시 테이블을 이용한 필터로 두 개의 해시 테이블과 두 개의 해시 함수를 제공하여 데이터 삽입, 삭제, 검색을 한다. Fig. 1은 쿠쿠 해싱함수의 알고리즘이다.

```
Algorithm : Insert Process Cuckoo hashing
1. Procedure Einsert(data)
2.   if  T[h1(data)] == data or T[h2(data)] == data then
3.       return;
4.   position <- h1(data);
5.   loop  n  times :
6.         if  T[position ] == NULL  then
7.                T[position ]  <-  data;
8.                data  <-  T[position] ;
9.               if position == h1(data) then
10.                        position <-  h2(data)
11.          else
12.               position <- h1(data);
13.          end if
14. rehash();
15. End Procedure
```

Fig. 1. Insert Process Cuckoo hashing algorithm

해시 테이블은 동일한 크기의 두 개의 작은 테이블로 분할되며 각각의 해시 함수는 두 테이블 중 하나의 테이블에 인덱스를 생성하고 삽입된 요소의 지문을 저장한다. 요소의 지문은 그 요소의 해시에서 파생된 비트 문자로 관리된다. 해시 테이블은 버킷의 배열로 구성되어 있으며, 버킷에 삽입된 요소는 두 해시 함수에 따라 수용 할 수 있는 두 개의 버킷으로 매핑 된다. 각 버킷은 지문의 변수를 저장하도록 설정할 수 있으며, 지문과 버킷의 크기로 식별된다. 쿠쿠 필터에 데이터를 삽입하기 위해 해시와 지문에 저장될 요소에서 두 개의 인덱스를 가져온다. 주어진 데이터를 해시 함수 h1(data) 을 사용하여 첫 번째 버킷의 위치를 계산하고 버킷의 빈 공간 슬롯에 데이터를 입력한다. 슬롯의 빈 공간이 없다면 두 번째 버킷에 h2(data) 함수를 호출하여 같은 과정을 반복한다. h1(data) 함수에서 빈 슬롯이 있을 경우에는 h2(data) 함수를 통한 탐색은 진행하지 않는다.

3. 제안 모델

제안모델은 중복 제거 시스템을 적용한 병렬 분산 스토리지 구축 시스템이다. 인라인 방식의 중복 제거는 데이터 저장 용량으로 생겨나는 서버의 부하 균형을 고려한 시스템이나 중복제거 처리가 데이터의 저장 시 병목현상이 될 가능성이 높다. 따라서 본 연구에서는 유사도를 적용하여 대용량 병렬 분산 스토리지를 구축할 수 있는 객체 스토리지에 인라인 방식의 중복 제거를 적응시키는 미들웨어 구축을 목적으로 한다.

3.1 시스템 아키텍처

제안된 시스템은 클라이언트 사용자에게 가상화 서비스를 제공하는 에이전트, 중복 제거된 데이터와 메타데이터를 저장하는 프로세스와 클러스터를 연동하는 매니저로 구성된다. 게이트 노드와 스토리지 노드 사이에 징검다리 시스템인 PD_SS(Parallel Distributed Stepping Stone Storage)를 배치하고 메타 데이터 베이스 서비스와 스토리지 노드를 분리하여 확장성을 용이하게 설계 했다. API(Application Programming Interface)는 Amazon S3를 호환하는 API를 제공하고 업로드 된 객체의 메타 데이터는 샤딩, 복제, 분산을 기준으로 데이터 서버 서비스를 구축한다. 스토리지에서는 한 대 이상의 노드를 클러스터로 구현하여 동기화를 시켰다.

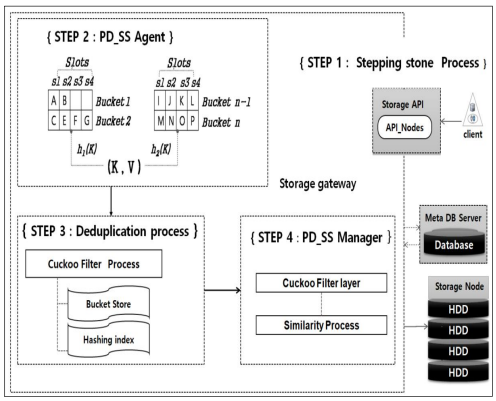

Fig.2. Architecture of PD_SS

Fig. 2는 데이터 중복 제거 병렬 분산 스토리지 구축을 기반으로 설계한 시스템의 전체 구조이다.

본 시스템의 처리 흐름은 다음과 같다.

Step 1. 스토리지 API를 통해 클라이언트 요청으로 데이터가 업로드 되면 PD_SS가 실행된다.

Step 2. 에이전트는 전송된 데이터를 해싱 엔진을 이용하여 지문인덱스를 생성한다. 해싱엔진은 쿠쿠 필터를 이용하여 삽입된 항목의 지문을 저장하는 쿠쿠 해시 테이블로 생성 후 데이터양의 증가에 대응하기 위해 수신한 데이터를 중복 데이터 배제 프로세서를 실행한다.

Step 3. 중복 데이터 배제 프로세서는 해시 인덱스와 버킷 스토어, 쿠쿠 필터 프로세스로 구성되어 클라이언트와 서버 간에 유사도가 높은 파일을 탐색한 후 매니저에게 결괏값을 전송한다.

Step 4. 매니저는 유사도 테이블, 쿠쿠 필터 레이어 계산 기능으로 구성되어 에이전트로부터 받은 데이터를 알고리즘을 이용하여 분산 스토리지 노드로 데이터를 전송하고 위치를 포함한 메타 데이터의 정보를 메타 데이터 베이스 서버에 저장한다.

3.2 중복 데이터 배제 프로세서

징검다리 에이전트는 쿠쿠 필터 프로세스와 중복데이터 배제 프로세스로 구성된다. 입력 데이터를 앵커기반의 유사도를 검사하고 쿠쿠 해싱의 탐색 알고리즘을 사용하여 중복 데이터 배제 프로세서를 실행하면서 모든 메타 데이터 정보를 징검다리 매니저의 테이블에 저장하도록 전송한다. 쿠쿠 필터는 삽입된 항목의 지문을 저장하는 쿠쿠 해시 테이블로 구성 된다. 항목의 지문은 해당 항목의 해시에서 파생 된 비트 문자열로 구성된다. 쿠쿠 해시 테이블은 해시 테이블의 삽입 할 항목을 두 개의 해시 함수로 버킷에 매핑 되는 버킷 배열로 관리되며 각각의 버킷은 가변 개수의 지문을 저장한다.

본 논문은 쿠쿠 해싱 알고리즘을 기본으로 멀티쓰레딩을 위한 공유 체크 섬 연산 호출을 알고리즘에 추가했다. 쿠쿠 필터는 지문 및 버킷 크기로 식별된다. 쿠쿠 해싱은 락 기반 해싱 방법으로서 각각의 버킷에 체크 섬인 SC를 하나씩 할당하는 방식을 사용한다. 그러나 쿠쿠 해싱 필터의 경우 두 개의 해시 테이블 중 첫 번째 해시 테이블에 데이터 삽입을 우선으로 하므로 해시 테이블 순환 호출 시 데이터 편중이 발생한다. 따라서 데이터 편중을 해결할 수 있도록 다중 노드 쿠쿠 필터를 설계하여 중복 제거 서버에 적용했다. 또한, 쿠쿠 필터를 다중 레이어로 구성할 경우 늘어나는 버킷의 수에 비례하여 비트가 기하급수적으로 증가함을 방지할 수 있고 단일 노드에서 다중 노드로 연산 되는 처리 시간 단축 효과를 기대할 수 있다.

Algorithm : Search Process Stepping stone hashing

```
1.  Procedure lookup( key )
2.     SC <- get_bucket_check(key, h1, h2);
3.     while :
4.       if SC then
5.           change( temp_data,
6.                   h1[INDEX01](key), h2[INDEX02](key));
7.           if INDEX01==h( rand_.key ) then
8.                   SC <- 0;
9.            end if
10.           INDEX02 <- h( rand_.key );
11.      else
12.          change( temp_data,
13.                  h1[INDEX01](key), h2[INDEX02](key));
14.          if INDEX02== h( rand_.key ) then
15.                  SC <- 1;
16.          end if
17.          INDEX01 <- h(rand_.key);
18.          return h1[INDEX].value;
19.       end if
20.          return KEY_NOT_FOUND;
21.    end while
22. End Procedure
```

Fig. 3. Stepping stone Search algorithm

Fig. 3은 본 논문에서 구현한 징검다리 검색 연산 알고리즘을 나타낸다. 중복 제거 서버의 쿠쿠 필터 프로세스는 다중 노드로 구성되고 레이어의 개수는 엔진에서 계산하여 전송한다. 각 버킷은 16바이트의 키와 16바이트의 밸류, 1바이트의 체크 섬인 SC로 구성되며 처음

해시 테이블이 만들어질 때 키와 밸류, SC는 널 값으로 초기화된다. 두 개의 해시 테이블에 대한 키에 해당하는 버킷이 존재하는지 검사한 후 값을 반환하는 작업을 수행한다. 버킷에 체크 섬인 SC를 획득하여 이미 값이 있다면 키를 받아 입력받은 값을 삭제한다.

중복 배제 프로세스는 저장할 파일의 계산된 유사도 해시 값들이 있는지 검색하고 유사도 해시 값이 있다면 메타정보를 매니저로 전송하고 유사도 해시 값이 없다면 대표 해시값을 계산한 다음 전송한다.

쿠쿠 해싱 프로세서의 레이어의 연산은 쿠쿠 해싱에 필요한 최적의 함수를 O(1 + N / B) = O(1) 식에 따라 계산한다. N은 데이터 수이고, B는 버킷 수이다. 데이터 수 N에 대한 버킷 수 B가 큰 경우 50개의 버킷에 200개의 데이터를 포함하게 될 경우 1개의 버킷리스트는 4개가 생성된다. 높은 확률로 연산 결과에 의해 두 개의 해시 테이블과 두 개의 해시 함수를 사용하여 충돌이 발생하면 기존에 저장되어있는 키 값은 다른 테이블로 이동하고 현재 계산된 값이 저장된다.

데이터 수 N에 대한 버킷 수 B가 작은 경우는 O(1 + N / B) = O(N)식에 대한 수식이 계산되어 50개의 버킷에 10,000개의 데이터를 포함할 경우 1개의 버킷에는 평균 200개의 데이터가 저장된다. 만일 해시 테이블의 용량이 M, 키가 N개 있을 때 M = 1.3 * N 일 경우 충돌과 공간의 낭비를 최소화할 수 있다고 판단하여 테이블의 75%가 채워지면 레이어를 추가한다.

단일 계층 쿠쿠 해싱 프로세스는 두 개의 함수로 해시테이블을 사용하기 때문에 데이터 삽입 과정에서 무한 반복이 발생한다. 반복으로 인한 충돌이 발생하게 되면 두 테이블의 크기를 모두 확장하고 새로운 해시 함수를 사용하여 모든 요소를 다시 삽입하여야 한다. 이에 비해 다중 계층 쿠쿠 해싱 프로세스는 각 레이어에 서로 다른 함수를 사용하여 해시 함수 결괏값의 충돌 발생 빈도를 줄일 수 있다.

3.3 클러스터링 및 중복 제거 기법

에이전트에서 전송받은 유사도를 기준으로 매니저는 클러스터 정보와 해시테이블의 연결 데이터를 기록, 저장하고 최적의 해싱 필터 레이어 개수를 결정한다.

레이어 값은 쿠쿠 해싱 프로세스에 전송되어 사용한다. 매니저는 쿠쿠 해싱 프로세스에 의해 새로운 버킷이 업데이트됐을 경우 중복된 버킷의 유무를 판단한다. 매니저는 사용자가 데이터를 업로드 하게 되면 쿠쿠 해싱 프로세스가 연동되어 해시 테이블과 지문을 저장하고 유사도 검사를 통해 에이전트로 받는 데이터를 유사도 테이블에 저장하고 쿠쿠 필터 레이어 계산 기능으로 구성된 정보를 저장한다. 유사도를 기준으로 중복 제거 서버 단위로 군집하고 분산 스토리지 노드로 데이터를 전송한다. 전송된 데이터는 위치를 포함한 메타 데이터의 정보로 메타 데이터 베이스 서버에 저장한다.

Algorithm : Process Stepping stone Agent

```
1.  Procedure SSD_Agent
2.        ht(h1,h2) <- ht(Data n)
3.        Cuckoo_HashValue <- Cuckoo_Hash(ht(h1,h2))
4.        Cuckoo_deputy <- sample(Cuckoo_HashValue)
5.        Send(Cuckoo_HashValue, CukooHash_Process)
6.       try :
7.         if TRUE then
8.              response <- 0
9.        while NOT response:
10.            Send(ht(h1,h2), SSD_Manager(response))
11.    except Error:
12.       return
13.  End Procedure
```

Algorithm : Process Cuckoo Hashing Filter

```
1.  Procedure CuckooHash_Process
2.        while :
3.          C_deputy <- get(SSD_Agent(n))
4.           while 1:
5.              Res <- compare(Similarity, SSD_Manager(m))
6.              if Res >max then
7.                    max <- Res
8.        Cuckoo_layer_cnt <- calculate(SSD_Manager(m))
9.        Response(SSD_Agent(n), SSD_Manager(m)))
10.       Send(Cuckoo_layer_cnt, SSD_Manager(m))
11. End Procedure
```

Algorithm : Process Stepping stone Manager

```
1.  Procedure SSD_Manager
2.         Similarity <- compare(C_deputy, HASH_INDEX)
3.         Send(Similarity, CukooHash_Process)
4.         Cuckoo_layer_cnt <- get(CukooHash_Process)
5.  End Procedure
```

Fig. 4. Stepping stone algorithm

Fig. 4는 징검다리 시스템의 전체적인 알고리즘을 나타낸다. 분산 스토리지 노드와 해시테이블의 모든 정보는 매니저의 테이블에 저장되어 관리되며 최적의 해싱 프로세스 레이어 개수를 결정하게 된다. 레이어 값을 에이전트의 중복 제거 프로세스인 쿠쿠 해싱 프로세스에 전송하여 사용한다. 쿠쿠 해싱 프로세스는 새로운 데이

터가 업데이트되면 해시 인덱스를 통해 중복 해시값을 판단한다.

4. 실험 및 평가

4.1 실험 환경

제안한 기법의 실험 환경 프로세서는 Intel Core i7-8700 CPU 3.20GHz 6 Core 12 Thread를 사용하고 CPU 2(4vCore), Red Hat Enterprise Linux 7.2의 플랫폼이 작동하는 16GB RAM과 512GB SSD를 탑재한 데스크 톱 PC에서 실험을 했다. 실험은 분산 스토리지 시스템에서 다중 계층 불룸필터를 적용한 중복제거 방법과 단일 쿠쿠 필터와 다중 쿠쿠 필터의 처리 성능을 측정했다. 유사도를 기반으로 한 클러스터링과 쿠쿠 필터가 적용된 제안 중복 기법에 대한 데이터 사이즈를 각각 청크와 버킷 사이즈 8KB로 비례하여 설정하고 입출력 저장 용량, 중복제거율, 프로세스 처리 속도를 측정하였다. 실험환경은 Table 1과 같다.

Table 1. Experimental environment

	Discribsion
CPU	Intel Core i7-8700 CPU 3.20 GHz
Memory	16GB
Storage Devides	512GB SSD * 8
OPerating System	Red Hat Enterprise Linux 7.2
HDFS 2.6.3 (master 1/slave 3)	Distributed File System (YARN prerequisite software)

4.2 결과 및 분석

실험 결과는 입력 용량 대비 중복률 및 배제율에 대한 실제 저장 장치가 사용한 중복제거율, 데이터 용량 비교와 처리 시간을 측정했다.

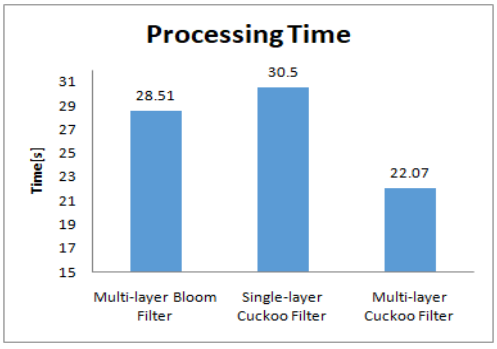

Fig. 5. Comparison of processing time by layer

Fig. 5는 다중 계층 블룸필터, 단일 계층 쿠쿠 필터, 제안된 다중 계층 쿠쿠 필터의 성능 비교를 나타낸다. 처리 시간은 병렬로 스레드를 처리할 수 있는 구조인 제안된 다중 계층 쿠쿠 필터가 6.44초 빠른 것으로 나타났다. 따라서 제안한 다중 계층 쿠쿠 필터는 20.2%의 성능 향상을 보였다.

Fig. 6은 제안된 중복 제거기법을 기준으로 데이터 입출력 용량에 대한 비교 및 분석을 하고, FIg. 7은 중복제거율을 비교 측정하며 Fig. 8은 처리 시간을 측정 후 프로세스의 성능을 비교 분석한다.

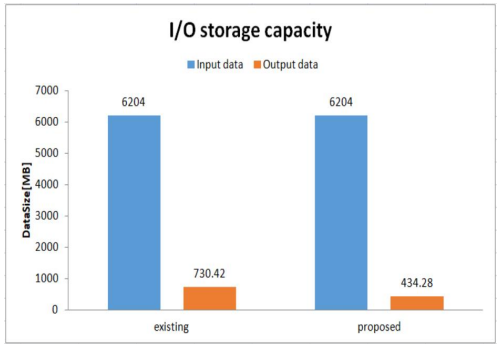

Fig. 6. I/O Storage capacity comparison

Fig. 6은 입출력 데이터의 스토리지 저장 용량 크기를 측정한 결과이다. 입력 데이터는 워크로드를 통해 6,204MB를 사용하고 출력 데이터를 저장한 스토리지는 징검다리 시스템의 중복 데이터 프로세스를 실행으로 중복데이터를 배제한 후 저장된 공간을 측정했다. 기존 블룸 필터의 다중 계층 레이어를 적용하는 분산 중복제거의 경우 730.42MB, 제한하는 방법은 434.28MB의 저장 용량을 사용했다. 블룸 필터의 다중 계층 레이어는 데이터가 동적으로 커지게 되면 블룸 필터의 변형이 이루어지고 중복 데이터 삭제 시에는 해싱된 모든 필터를 재구축 할 시간과 메모리가 동반되었다. 하지만 제안된 논문은 삽입, 삭제, 검색에 대한 용이성을 가지고 쿠쿠 필터 프로세서의 유사도를 기반으로 수용할 수 있는 두 개의 인덱스 중 하나의 지문을 확인한 후 해당 지문이 존재한다면 쉽게 테이블에서 요소를 제거하기 때문에 노드 간의 중복 데이터 비율이 높아진다.

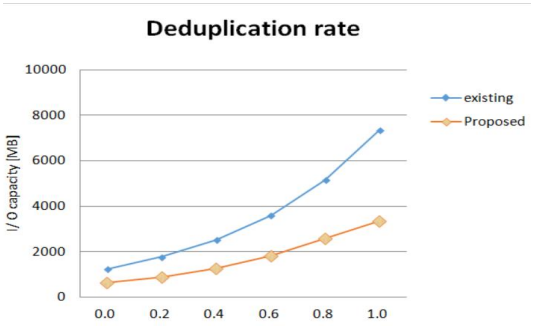

Fig. 7. Deduplication rate measurement result

Fig. 7은 각각의 중복률에서 배제율과 데이터 사이즈를 파라미터로 나타낸다. 중복률과 데이터 사이즈는 동일한 중복률에 대해서, 데이터 사이즈가 커짐에 따라 배제율은 급속히 저하하는 것을 알 수 있다. 동일한 중복률에 대해서는 데이터 사이즈가 작을수록 배제 수가 증대하는 것을 알 수 있다. 데이터 크기가 작을수록 배제율은 높아지는 동시에 배제된 데이터 수는 증대한다. 제안된 시스템으로 중복률을 검사한 결과 현저하게 낮게 나타난 것을 볼 수 있다.

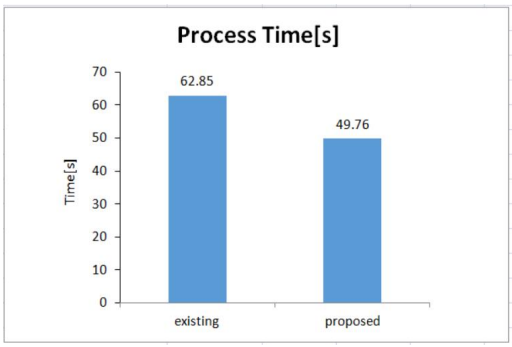

Fig. 8. Processing time measurement result

Fig. 8은 징검다리 시스템을 이용한 처리 시간을 나타낸다. 기존 블룸 필터의 다중 계층 레이어는 6,204MB 데이터를 중복 제거하는데 62.85초가 소모되고 제안된 시스템은 49.76초로 나타났다. 입력과 출력 및 탐색 모두 동일한 메모리에 실험마다 200번 반복하여 평균을 기록하였다.

5. 결론

본 논문은 소프트웨어가 내장된 스토리지 가상 머신의 유사도 기반 클러스터링과 다중 계층 쿠쿠 해싱 필터를 활용한 분산중복 제거 프로세스 실행 기법을 제안하였다. 비정형 데이터인 대체 데이터의 처리를 빅 데이터 플랫폼에서 탐색 공간 최소화를 통한 분석 성능 향상 시스템이 구현되어 있지만, 보다 많은 실시간으로 발생하는 데이터를 지연 없이 적재해야 하며 서로 다른 스토리지의 중복 데이터에 대해 데이터 중복 제거 시스템을 사용하여 스토리지 공간을 효율적으로 관리할 필요가 있다. 제안 방법은 입력된 데이터를 쿠쿠 해시 테이블로 지문과 요소를 버킷에 저장하고 쿠쿠 해시 알고리즘을 이용하여 유사도를 측정하고 다중 계층 쿠쿠 필터를 이용하여 인덱스 조회를 병렬 처리한다. 실험 결과로는 기존 블룸 필터의 다중 계층 레이어보다 중복제거율은 10.3% 향상하고 처리 시간은 8.9% 단축시키는 결과를 보였다. 그러나 데이터의 단일 크기가 작은 파일로 한정되어 있으며 복수 파일의 백업 등은 다른 결과가 예상되므로 더욱 광범위한 조사를 시행할 필요가 있다. 추가로 다양한 크기의 데이터를 일괄 처리하는 성능 비교가 필요하다.

REFERENCES

[1] L. Richard et al. (2008). Emerging Tech and Modern IT: *The Key to Unlocking Your Data Capital*, (Online). http://www.idc.com

[2] Swift, (n. d.). *OpenStack Object Storage*.(Online).https://docs.openstack.org/swift/latest/.

[3] A. Sage et al. (2006). Ceph: A Scalable, *High-Performance Distributed File System. OSDI,* 307-320

[4] Leo Project. (2014). *The Lion of Sorage Systems. LeoFS.* (Online). http://leo-project.net/.

[5] P. Raj & A. Raman. (2018). *Software-defined storage (SDS) for storage virtualization.* In Software-defined cloud centers (pp. 35-64). Springer, Cham.

[6] Brodkin et al. (2018). *EMC Atoms Cloud Storage.* (Online). http://www.emc.com/storage/atmos/atmos.htm/.

[7] Amplidata.(2020). *Himaraya.* (Online).http://amplidata.com/.

[8] Amazon. (n. d.). *Amazon simple storage service (amazon s3).* (Online).http://aws.amazon.com/s3/

[9] Google. (n. d.). *Google cloud storage.* (Online). https://cloud.google.com/storage/docs/json_api/v

1/objects.

[10] X. Zhao et al. (2014). A scalable deduplication file system for virtual machine images. Parallel and Distributed Systems, *IEEE Transactions, 25(5),* 1257-1266,
DOI : 10.1109 / TPDS.2013.173

[11] R. Kutzelnigg. (2010). An improved version of cuckoo hashing: Average case analysis of construction cost and search operations, *Math. Comput. Sci., 3(1),* 47-60.

[12] D. Yoon & D. H. Kim. (2018). Distributed data deduplication technique using similarity based clustering and multi-layer bloom filter. *Journal of Korean Institute of Next Generation Computing, 14(5),* 60-70.

[13] S. S. Nam & C. H. Seo. (2016). Privacy Preserving Source Based Deduplicaton Method. *Journal of Digital Convergence, 14(2),* 175-181
DOI : 10.14400/JDC.2016.14.2.175

[14] S. W. Jeong et al. (2018). *Cyber KillChain Based Security Policy Utilizing Hash for Internet of Things. Journal of Digital Convergence, 16(9),* 179-185.
DOI : 10.14400/JDC.2018.16.9.179

[15] Y. S. Jeong et al. (2015). An Efficient data management Scheme for Hierarchical Multi-processing using Double Hash Chain. *Journal of Digital Convergence, 13(10),* 271-278.
DOI : 10.14400/JDC.2015.13.10.271

[16] Y. S. Jeong et al (2015). Multi-Attribute based on Data Management Scheme in Big Data Environment. *Journal of Digital Convergence, 13(1), 263-268*
DOI : 10.14400/JDC.2015.13.1.263

[17] R. Rivest. (1992). *The MD5 Message-Digest Algorithm* , 1992RFC, IETF Network Working Group.

김 영 아(Yeong-A Kim) [정회원]

· 2019년 8월 : 경남과학기술대학교 컴퓨터공학과(공학석사)
· 2017년 ~ 현재 : ㈜ En-Core HRD 연구원
· 관심분야 : 빅데이터, 정보보안, 네트워크, 인공지능, ICT, IOT
· E-Mail : finish07@encore.com

김 계 희(Gea-HeeKim) [정회원]

· 2013년 8월 : 한국방송통신대학교 정보과학과(이학석사)
· 2017년 2월 : 경남과학기술대학교 컴퓨터공학과(공학박사) ·
· 관심분야 : Vanet, IoT, 무선네트워크, 차량 간통신
· E-Mail : jenni7@naver.com

김 현 주(Hyun-Ju Kim) [정회원]

· 1988년 2월 : 경상대학교 컴퓨터과학(이학사)
· 1990년 8월 : 숭실대학교 컴퓨터공학(공학석사)
· 2000년 8월 : 경상대학교 컴퓨터과학(이학박사)
· 2002년 ~ 현재 : 경남과학기술대학교 컴퓨터공학과 교수
· 관심분야 : XML, 정보검색, 데이터마이닝
· E-Mail : khj@gntech.ac.kr

김 창 근(Chang-Geun Kim) [정회원]

· 1990년 2월 : 경남대학교 컴퓨터공학과(공학석사)
· 1999년 2월 : 경남대학교 컴퓨터공학과(공학박사)
· 1995년 3월 ~현재 : 경남과학기술대학교 교수
· 관심분야 : 컴퓨터네트워크, 데이터통신, e-비즈니스
· E-Mail : cgkim@gntech.ac.kr