Journal of KIIT. Vol. 17, No. 2, pp. 13-20 Feb. 28, 2019. pISSN 1598-8619, eISSN 2093-7571 13
http://dx.doi.org/10.14801/jkiit.2019.17.2.13

빅데이터 환경에서의 효율적인 동적 블룸 필터 기법

박홍규*, 박남훈**

Efficient Dynamic Bloom Filter Method in Big Data Environment

Hongkyu Park*, Nam Hun Park**

"이 논문은 동양미래대학교 학술연구지원사업의 지원으로 수행된 연구과제입니다."

요 약

최근 빅데이터 산업의 규모가 커지면서 더 방대한 양의 데이터 처리, 실시간 데이터 분석 등 산업의 요구사항들도 점차 고도화되고 있다. 많은 빅데이터 분석 플랫폼에서 탐색 공간 최소화를 통한 분석 성능 향상을 위하여 블룸 필터를 많이 활용하지만, 기존의 방법들은 시스템 요구사항이나 빅데이터의 특성들을 잘 반영하지 못하고 있으며, 주로 긍정오류율을 줄이는 데 초점을 맞추고 있다. 하지만 빅데이터 영역에서는 긍정오류율보다 실시간으로 발생되는 방대한 양의 데이터를 지연 없이 적재해야 하며, Spark, Storm과 같은 인-메모리 처리 엔진의 경우에는 저장 공간 또한 효율적으로 활용해야 한다. 본 논문에서는 이러한 빅데이터의 특성을 고려하여 적재 성능과 저장 공간 비용을 최적화한 동적 블룸 필터 기법들을 제안하고, 실험을 통해 이들의 효율성과 성능을 검증한다. 본 연구 결과를 블룸 필터를 이미 활용 중인 빅데이터 분석 플랫폼에 적용하여 플랫폼의 성능 향상에 기여할 수 있을 것으로 기대된다.

Abstract

Recently the scale of the big data industry has grown, industry requirements such as bigger data processing and real time data analysis are gradually becoming sophisticated. For this reason, many big data analysis platforms utilize Bloom Filter to improve analytical performance through minimization of search space, but previous methods focus mainly on reducing false positive errors and do not well reflect the requirements of system requirements and the characteristics of big data. However, in the big data area, it is necessary to load a huge amount of data occurring in real time without delay, and in the case of an in-memory processing engine such as Spark, Storm, storage capacity must also be utilized efficiently. We propose a dynamic bloom filter methods optimizing insertion performance and storage efficiency by considering those characteristics and verify these efficiency and performance through experiments. By applying the proposed methods to big data analysis platforms which use bloom filters, we expect to improve the performance of those platforms.

Keywords
big data, bloom filter, dynumic bloom filter, false positive error

* 동양미래대학교 컴퓨터소프트웨어공학과
- ORCID: https://orcid.org/0000-0002-6446-9081
** 안양대학교(교신저자)
- ORCID: https://orcid.org/0000-0002-3716-5760

· Received: Jan. 16, 2019, Revised: Jan. 31, 2019, Accepted: Feb. 03, 2019.
· Corresponding Author: Nam Hun Park
Dept of Computer Science, Anyang University, Kyunggi, Korea,
Tel. +82-32-930-6021, Email: nmhnpark@anyang.ac.kr

Ⅰ. 서 론

빅데이터는 인공지능, IoT(사물인터넷), 생명공학 기술 등 정보통신기술(ICT)의 융합으로 이루어지는 차세대 산업혁명인 4차 산업혁명의 핵심 기반 기술이다. 스마트 팩토리, 스마트 팜, 웹서버 로그 등 다양한 센서들로부터 발생되는 방대한 양의 빅데이터를 효과적으로 처리/분석하여 자동화 또는 개인화 서비스를 하기 위해서는 실시간 빅데이터 처리 및 분석이 필수적이다. 방대한 양의 빅데이터를 하나의 서버를 증설하는 방식(Scale-up)으로 처리 용량을 증설하는 것은 비용적인 면이나 확장성 측면에서 한계가 존재하므로, 대부분의 빅데이터 처리 기술은 분산 시스템을 기반으로 한다. 빅데이터 초창기에는 맵-리듀스(Map-Reduce)와 하둡 분산 파일 시스템(HDFS)으로 구성된 하둡(Hadoop)[1] 분산 시스템을 중심으로 발전하였다. 하지만 디스크 기반 처리로 인한 느린 성능과 단일 네임노드로 인한 SPOF (Single Point Of Failure) 등의 문제들로 인해, 최근에는 Apache Spark[2], Storm[3], BigStream[4] 등 실시간 빅데이터 분석을 위한 메모리 기반의 분산 시스템들이 더 활발히 개발되고 연구되고 있다.

블룸 필터(Bloom Filter)[5]는 원소가 집합에 속하는지 여부를 검사하는데 사용되는 확률적 자료 구조로써, 데이터베이스 연산 최적화[6][7], 네트워크 라우팅 최적화[8][9], 침입 탐지[10][11] 등 많은 분야에서 탐색 공간 최소화를 통한 성능 개선을 위하여 많이 활용되었다. 원소들이 집합에 속해있는지 여부를 저장하는 비트테이블이 존재하고, 해쉬 함수를 이용하여 원소의 값에 대항하는 비트를 1로 갱신한다. 블룸 필터는 집합 내 원소의 개수(카디널리티), 긍정오류율, 해시 함수의 개수, 비트테이블의 크기가 서로 영향을 미친다. 데이터의 카디널리티와 긍정오류율이 주어지면, 주어진 긍정오류율 이하를 보장하기 위하여 비트테이블의 크기와 해시 함수의 개수가 정해진다. 하지만 데이터의 특성을 미리 알 수 없거나 센서 스트림과 같이 데이터가 계속해서 발생되는 응용분야에서는 긍정오류율 보장이 어렵다. 왜냐하면 처음에 설정된 카디널리티보다 실제 데이터의 카디널리티가 더 크게 되면 긍정오류율을 만족시키기 위해서는 비트테이블의 크기를 확장해야하기 때문이다. 비트 테이블의 확장 연산은 기존의 모든 데이터들을 대상으로 재구성 연산을 수행하여야 하므로 연산 비용이 매우 크다. 이러한 단점을 개선하기 위하여 동적 블룸 필터(DBF, Dynamic Bloom Filter)[12]는 긍정오류율의 조건을 만족시키면서 동적으로 블룸 필터를 확장시키는 방법을 제안한다. 실제 데이터의 카디널리티가 주어진 카디널리티 값보다 커지면 블룸 필터를 재구성하는 대신, 기존의 1차원 비트테이블을 N차원의 비트테이블로 표현함으로써, 블룸 필터를 동적으로 확장하는 방식이다. 하지만 DBF[12]는 데이터 적재와 탐색 비용이 O(N×k)이므로, O(k)인 정적 블룸 필터(SBF, Static Bloom Filter)[5]에 비해 데이터 삽입 및 탐색 비용이 많이 드는 단점이 존재한다.

스마트 팩토리, 스마트 팜, 스마트 그리드와 같은 IoT 분야나 대용량 로그 분석 등 빅데이터의 주요 응용 분야에서 발생하는 데이터들은 데이터의 특성(카디널리티, 데이터 분포)을 미리 알 수 없으며, 한 번 발생된 데이터는 변경되지 않으며, 발생 시간의 순서대로 데이터가 적재된다. 본 논문은 이와 같은 빅데이터 응용 분야의 특성들을 고려하여 DBF 기반의 효율적인 블룸 필터 방법을 제안한다. 기존의 빅데이터 처리 플랫폼[13][14]에서도 탐색 공간 최소화를 통한 성능 향상을 위하여 블룸 필터를 활용하지만 SBF[5]를 활용하여 긍정오류율을 최소화하는 것이 주목적이다. 하지만 빅데이터 분야에서는 긍정오류율보다 실시간으로 발생되는 방대한 양의 데이터를 지연 없이 적재해야 하며, Spark[2], Storm[3]과 같은 인-메모리 처리 엔진의 경우에는 저장 공간 또한 효율적으로 활용하여야 하므로, 적재 성능과 저장 효율성이 고려되어야 한다. 하지만 적재 성능과 저장 효율성은 trade-off 관계이므로 본 논문에서는 적재 성능 중심의 블룸 필터 방식(b-DBF)과 저장 효율성 중심의 블룸 필터 방식(s-DBF)을 제안한다. 많은 빅데이터 분석 플랫폼[2]-[4][13][14]에서 데이터의 효율적인 저장과 처리를 위하여 이미 블룸 필터 기법들을 활용 중이므로 본 연구 결과를 기존 플랫폼에 쉽게 적용할 수 있다. 이를 통해 빅데이터 플랫폼의 성능 향상에 많은 영향을 끼칠 수 있을 것으로 기대한다.

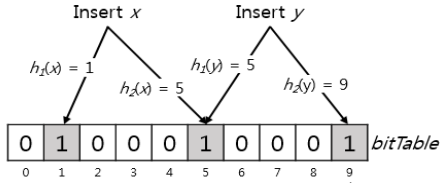
II. 관련 연구

SBF[5]는 그림 1(a)와 같이 집합 $X = \{x_1, x_2, ...x_n\}$의 항목들의 소속 여부를 m 비트로 이루어진 비트테이블에 저장하고, 항목의 집합 소속 여부를 검사하는데 사용되는 확률적 자료 구조이다. 비트테이블은 초기에는 모두 0으로 표기되며, X의 각각의 항목에 대해 해시의 결과값이 $\{1,2,...,m\}$인 k개의 해시 함수(h_1, h_2, h_k)를 적용하여 해당하는 비트테이블의 비트를 1로 갱신한다. 예를 들어, 크기가 10인 비트테이블이 존재하면, 어떤 원소의 값을 해싱 함수를 통해서 1~10까지의 값으로 변환한 후, 해당 자리의 비트를 1로 설정한다. 이를 통해 어떤 원소가 주어졌을 때, 해당 원소가 집합에 속해 있는지 여부를 판단할 수 있다. 블룸 필터에 의해 어떤 원소가 속해 있지 않은데, 속해있다고 판단하는 긍정오류(False Positive Error)[5]는 존재하지만, 반대로 어떤 원소가 속해있는데 속해있지 않다고 판단하는 부정 오류(False Negative Error)는 존재하지 않는다. 블룸 필터는 집합 내의 항목의 개수(카디널리티, *n*), 긍정오류율(*p*), 해쉬 함수의 개수(*k*), 비트테이블의 크기(*m*)들을 모두 정의가능하며, 집합 내 원소의 숫자가 증가할수록, 비트테이블의 크기가 작을수록, 해시 함수의 개수가 작을수록, 긍정오류율도 증가한다. 긍정오류율(*p*)은 아래와 같이 계산될 수 있다[5].

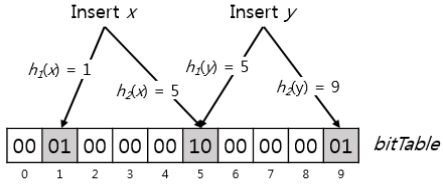
$$p \approx (1-(1-\frac{1}{m})^{kn})^k \approx (1-e^{-\frac{kn}{m}})^k \quad (1)$$

카운팅 블룸 필터[15]는 원소 삭제가 불가능한 블룸 필터의 문제점을 보완하기 위해서 고안되었다. 블룸 필터는 원소의 소속 여부를 1 비트로 표현하므로, 해당 원소가 몇 개가 속해있는지를 알 수 없어 데이터 삭제가 불가능한 반면, 카운팅 블룸 필터는 그림 1(b)와 같이 X 비트를 사용하여 원소의 개수도 표현함으로써 원소의 삭제가 가능한 반면, [5]에 비해 더 많은 저장 공간이 필요하다.

SBF는 집합 내 원소의 개수(카디널리티), 긍정오류율 등 정해진 설정값에 따라 비트테이블의 크기가 정해진다.



(a) 정적 블룸 필터[5]
(a) Static bloom filter[5]



(b) 카운팅 블룸 필터[12]
(b) Counting bloom filter[12]

그림 1. 정적 블룸 필터와 카운팅 블룸 필터
Fig. 1. Static bloom filter and counting bloom filter

처음에 정의된 카디널리티와 실제 카디널리티의 차이가 발생하면 긍정오류율의 조건을 만족시키기 위해서는 비트테이블의 크기를 확장해야 하며, 이는 기존의 모든 데이터들을 대상으로 재구성 연산을 수행해야 하므로 연산 비용이 매우 크다. 그러므로 데이터의 특성을 미리 알 수 없는 응용분야에서도 블룸 필터를 효과적으로 활용할 수 있도록 DBF[12]가 제안되었다. DBF는 기존의 1개의 1차원 비트테이블 대신 N개의 비트테이블로 표현함으로써, 블룸 필터를 동적으로 확장하는 방식이다. 블룸 필터를 갱신하면서 현재까지 저장된 데이터들의 카디널리티를 함께 갱신한다. 갱신된 카디널리티가 한계치(Threshold)를 넘어서면 신규 블룸 필터를 생성하는 방식으로 비트테이블을 N개로 확장한다. 각 블룸 필터는 카운팅 블룸 필터[15]를 기반으로 구현되어 있다.

III. 빅데이터 환경에서의 DBF 기법

3.1 기본 동적 블룸 필터(b-DBF)

본 논문에서는 한 번 발생된 데이터는 변경되지 않으며, 발생 시간의 순서대로 데이터가 발생되는 빅데이터의 특성을 활용하여 SBF[5] 기반의 DBF 방법을 제안한다. 이를 기본 동적 블룸 필터(b-DBF,

basic DBF)라고 명명한다.

b-DBF는 여러 개의 정적 블룸 필터(SBF_i, 1<=i<=N)로 구성된다. 구성 초기에는 하나의 SBF를 가지며, SBF의 비트테이블의 크기는 설정변수(긍정오류율(p), SBF가 가질 수 있는 값의 개수(=카디널리티, n), 해시 함수의 개수(k))들의 값에 의해 결정되며, 주어진 긍정오류율(p)을 보장할 수 있는 비트테이블의 크기가 결정된다. 만약 데이터셋의 카디널리티가 주어진 카디널리티(n)보다 큰 경우가 발생하면, b-DBF는 새로운 SBF_{i+1}를 할당한다. 이와 같은 방식으로 그림 2와 같이 b-DBF는 데이터를 적재하면서 여러 개의 SBF를 가진 형태로 확장된다.

[12]에서 제안한 DBF와 1차원 블룸 필터를 여러 개로 확장하는 방식은 동일하나, [12]는 카운팅 블룸 필터[12] 기반으로 구현되어 있으나, b-DBF는 SBF를 기반으로 구현되어 있다. 이로 인해 [12]는 데이터 삭제가 가능한 반면, 데이터 삽입 시에 현재 삽입하고자 하는 데이터를 저장할 수 있는 블룸 필터를 찾기 위하여 DBF를 구성하고 모든 1차원 블룸 필터를 탐색하여야 하므로, 복잡도가 O(N+k)이다. 여기에서 N은 DBF를 구성하고 있는 블룸 필터의 개수이며, k는 해시함수의 개수이다. 반면 b-DBF는 데이터 삭제가 불가능한 반면, 데이터 삽입 시 마지막 블룸 필터에 삽입하거나 마지막 블룸 필터가 꽉 찬 경우, 즉 마지막 블룸 필터가 저장하고 있는 값의 개수가 카디널리티(n)와 같은 경우에는 새로운 블룸 필터를 할당하여 사용하므로, 연산의 복잡도는 O(k)이다. 삽입 연산의 의사코드는 그림 3과 같다. 서론에서 언급한 바와 같이 대부분의 빅데이터 응용분야에서의 데이터들은 시간의 순서대로 발생하며, 한 번 저장된 데이터가 변경되지 않지 않는 특징이 있으므로 [12]보다 d-DBF가 더 적합하다.

데이터 집합 내에 항목 x의 존재 여부를 판단하는 멤버십 질의(Membership Query)는 그림 4와 같다. x의 존재 여부를 판단하기 위해서는 b-DBF를 구성하고 있는 모든 1차원 블룸 필터를 탐색하여야 한다. 각 블룸 필터를 대상으로 k개의 해시함수의 결과값 ($h_1(x)$, $h_2(x)$,…,$h_k(x)$)에 해당하는 모든 비트가 1로 되어 있다면, x는 존재한다고 판단한다.

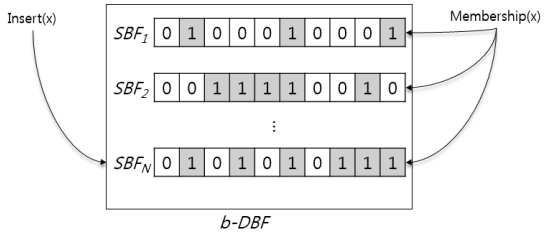


그림 2 기본 동적 블룸 필터
Fig. 2. b-DBF

Algorithm 1 Insert (x)

```
n : the number of items in a set
p : false positive error ratio
k : the number of hash functions
BF : Dynamic Bloom Filter
N : the number of bitTables
BF_i : i^th bitTable of BF
BF_i.c : The number of items of BF_i
Require: x is not null
1: counter = 0
2: ActiveBF ← GetActiveBF()
3: if ActiveBF is null then
4:   ActiveBF ← CreateNewBF(n, k, p)
5:   Add ActiveBF to this dynamic Bloom filter.
6:   N ← N + 1
7: for i = 1 to k do
8:   if ActiveBF[hash_i(x)] == 0 then
9:     ActiveBF[hash_i(x)] = 1
10:  else
11:    counter <- counter+1
12: if counter != k then
13:   ActiveBF.c <- ActiveBF.c + 1
GetActiveBF()
1:   if BF_N.c < n then
3:     Return BF_N
4:   Return null
```

그림 3 기본 동적 블룸 필터의 삽입 연산
Fig. 3. Insert operation of b-DBF

Algorithm 2 Membership (x)

```
Require: x is not null
1: for i = 1 to N do
2: counter ← 0
3:    for j = 1 to k do
4:       if BFi[hashj(x)] = 0 then
5:          break
6:       else
7:          counter ← counter + 1
8:    if counter = k then
9:       Return true
10: Return false
```

그림 4. 기본 동적 블룸 필터의 멤버쉽 연산
Fig. 4. Membership operation of b-DBF

3.2 저장 효율성 기반 DBF

b-DBF와 [12]는 데이터 x를 삽입할 때 x가 이미 블룸 필터에 저장되어 있는지 여부를 검사하지 않아 x가 여러 1차원 블룸 필터(SBF_i)에 저장될 수 있으므로 전체 데이터셋의 분포도에 따라 저장공간 사용율이 달라진다.

저장 공간 기반 동적 블룸 필터(s-DBF, storage-focused DBF)는 b-DBF와 [12]가 가지고 있는 저장 공간 사용의 비효율성을 제거한다. s-DBF는 b-DBF와 마찬가지로 여러 개의 정적 블룸 필터로 구성된다. 그림 5와 같이 s-DBF는 데이터 삽입 시 해당 데이터의 삽입 이력을 확인한 후, 삽입된 이력이 없는 경우에만 b-DBF와 같은 방식으로 데이터 삽입을 수행하며, 삽입된 이력이 있는 경우에는 삽입 연산이 종료된다. 그러므로 s-DBF은 항목 x는 오직 하나의 SBF_i에만 저장하는 것을 보장하므로, 저장 공간을 낭비하지 않고 효율적으로 활용한다. 하지만 데이터 삽입 시, 항목 x의 삽입 이력을 확인하여야 하므로 삽입 연산의 성능은 b-DBF에 비해 좋지 않다. 각 알고리즘별 성능 및 주요 차이는 표 1과 같다.

Algorithm 3 Insert (x)

```
Require: x is not null
1: ActiveBF ← GetActiveBF(x)
2: if ActiveBF is not null then
3:   for i = 1 to k do
4:     ActiveBF[hash_i(x)] = 1
5:   ActiveBF.c <- ActiveBF.c + 1
GetActiveBF(x)
1: for i = 1 to N do
2:    counter = 0
3:    for j = 1 to k do
4:       if BF_i[hash_j(x)] == 0 then
5:          break
6:       else
7:          counter ← counter + 1
8:    if counter == k then //if x exists in BF_i
9:       return NULL
10: if BF_N.c < n then
11:   Return BF_N
12: else
13:   ActiveBF ← CreateNewBF(n, k, p)
14:   Add ActiveBF to BF
15:  N ← N + 1
16:  return ActiveBF
```

그림 5. 기반 동적 블룸 필터의 삽입 연산
Fig. 5. Insert operation of s-DBF

표 1. 각 블룸 필터 별 비교
Table 1. Comparison of bloom filter methods

	SBF[5]	DBF[6]	b-DBF	s-DBF
Insert cost	O(k)	O(N + k)	O(k)	O(N + k)
Membership cost	O(k)	O(N + k)	O(N + k)	O(N + k)
Storage	Static allocation	- Dynamic allocation - Usage varies according to data distribution - It implements based on CBF[12]. So storage usage is relatively high.	- Dynamic allocation - Storage usage varies according to data distribution	- Dynamic allocation - Efficient storage usage irrespective of data distribution
Date Delete	X	O	O	X
Data characteristics of applications	data[1]	1) Dynamic data[2] 2) Requires data delete and update	1) Dynamic data[2] 2) Append only data(No data update) 3) In case of data insert operation with delay	1) Dynamic data[3] 2) Append only data(No data update) 3) Critical storage usage (e.g. in-memory database/platform)

1) Known data characteristics(e.g. cardinality) before building bloom filter
2) Unknown data characteristics(e.g. cardinality) before building bloom filter
3) Only the deletion with time condition is possible

Ⅳ. 성능 평가

4.1 실험 환경

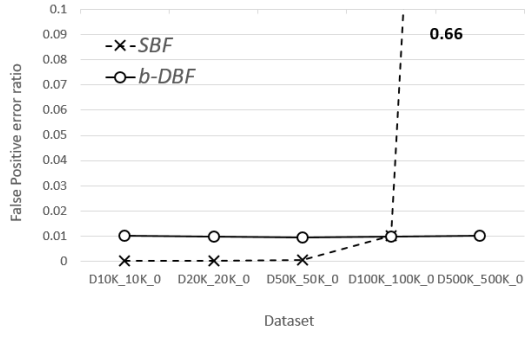
4장에서는 본 논문에서 제안한 b-DBF와 s-DBF를 SBF[5], DBF[12]와의 비교 실험을 통해 성능을 확인한다. 실험 환경은 아마존 클라우드 서비스 AWS의 t2.xlarge 인스턴스(16GB, 4 vCPU)에 진행되었다.

본 논문에서 제안한 블룸 필터 기법들은 데이터셋의 카디널리티, 값의 분포 등의 특성에 따라 성능의 차이가 존재하므로, 이를 확인하기 위하여 데이터생성기를 구현하였다. 데이터 생성기는 카디널리티(c), 전체 데이터셋의 크기(s)와 zipf value(z)를 인자를 받아서 데이터를 생성한다. 데이터 생성기는 Zipfian 분포를 따르는 데이터셋을 생성하며, zipf value(z)를 통해 데이터의 불균형 정도(Skewness)를 조절한다. 만약 z의 값이 0이면 정규 분포의 데이터셋을 생성하며, z의 값이 4라면 매우 편향된(Highly-skewed) 데이터셋을 생성한다. 데이터셋을 D*$c*_*$s*_*$z*로 표기한다. 예를 들어, D100_1000_0은 [1,100]의 값으로 1,000개의 데이터를 생성하며, 이 데이터는 정규 분포를 따른다.

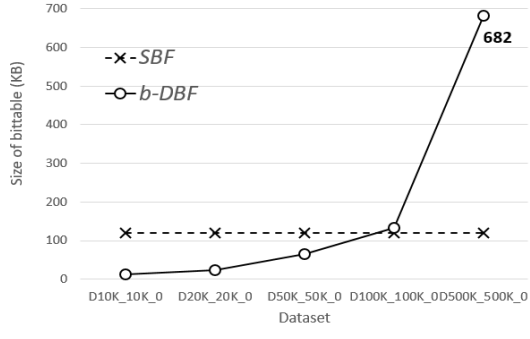
4.2 성능 평가

그림 6은 데이터셋의 크기에 따른 SBF와 b-DBF 기법의 긍정오류율과 저장 공간을 비교한 실험이다. 긍정오류율(p)은 0.01, 해시 함수의 개수는 7개로 설정하였으며, SBF는 카디널리티를 100K로 가정하고 비트테이블을 구성하였다. 6(a)에서 보는 바와 같이 SBF는 데이터셋의 크기가 100K까지는 긍정오류율이 0.01 이하로 유지되다가, 500K가 되면 0.66으로 급격히 증가하게 된다. 반면, b-DBF는 데이터의 크기가 커지더라도, 정의된 긍정오류율을 넘지 않는다. 하지만 6(b)와 같이 데이터셋이 커짐에 따라 비트 테이블이 확장되므로 저장 공간이 늘어난다.

그림 7과 8은 DBF, b-DBF와 s-DBF의 적재 성능과 저장 공간을 비교한 실험이며, 데이터셋의 카디널리티와 z 값은 각각 100K와 0으로 설정한 데이터셋을 사용하였으며, 적재 성능은 초당 처리량, 즉 초당 삽입된 레코드의 개수로 측정하였다. 세 알고리즘 모두 데이터의 크기가 커짐에 따라 SBF들의 개수도 늘어나게 된다. DBF와 s-DBF는 데이터를 삽입할 때 SBF들을 모두 탐색해야 하므로, 데이터의 크기가 커짐에 따라 적재 성능이 조금씩 저하되는 것을 볼 수 있다. 반면, b-DBF는 SBF의 탐색 없이, 마지막 SBF에 삽입하면 되기 때문에 데이터 크기가 커짐에 따른 성능 저하는 없는 것을 알 수 있다.



(a) 긍정오류율
(a) False positive error ratio



(b) 비트테이블의 크기
(b) Size of bittable

그림 6. 정적 볼룸 필터와 기본 동적 블룸 필터의 비교 실험
Fig. 6. Comparison of SBF and b-DBF

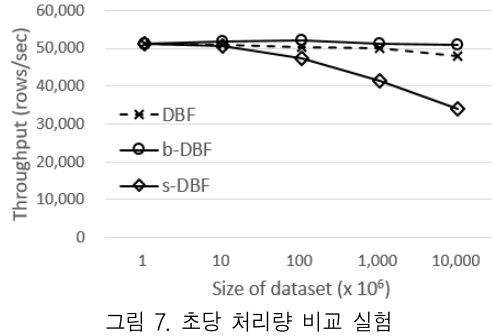


그림 7. 초당 처리량 비교 실험
Fig. 7. Experimental comparison of throughput

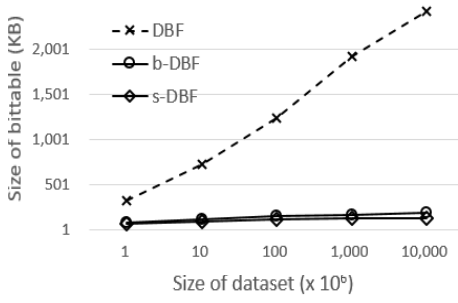


그림 8. 저장 효율성 실험 (k=5, p=0.01)
Fig. 8. Experiment for storage efficiency (k=5, p=0.01)

그림 8에서 볼 수 있듯이, DBF[12]는 CBF[15]를 기반으로 구현되어 있으므로, b-DBF와 s-DBF에 비해서 많은 저장 공간을 사용하며, s-DBF는 특정 항목이 하나의 SBF에만 저장됨을 보장하기 때문에 b-DBF에 비해 적은 저장 공간을 사용한다.

V. 결론 및 향후 과제

최근 많은 분야에서 빅데이터의 실시간 처리 및 분석을 위하여 블룸 필터를 많이 활용하지만 시스템 요구사항이나 빅데이터의 특성들을 잘 반영하지 못하고 있다. 빅데이터를 위한 블룸 필터는 방대한 탐색 공간을 최소화하기 위하여 사용되지만, 그보다 실시간으로 발생되는 방대한 양의 데이터를 지연 없이 적재해야 하며, 인-메모리 처리 엔진[2][3]의 경우에는 저장 공간 또한 효율적으로 활용하여야 한다. 그러므로 본 논문에서는 블룸 필터의 삽입 비용과 저장 비용을 고려한 DBF 기법들을 제안한다. 방대한 양의 데이터가 계속적으로 발생되고 카디널리티와 같은 데이터의 특성을 미리 규정지을 수 없는 특성을 고려하여, DBF로 구현하였다. 기존의 DBF[12]는 데이터의 삭제를 위하여 CBF[15]를 기반으로 구현되었지만, 데이터의 삭제가 변경이 거의 발생하지 않는 빅데이터의 특성을 고려하여 본 논문은 SBF[5] 기반으로 구현하여 [12]에 비해 저장 효율성 측면에서 우세하다. 많은 빅데이터 분석 플랫폼[2]-[4][3][4]에서 데이터의 효율적인 저장과 처리를 위하여 이미 블룸 필터 기법을 활용 중이므로 본 연구 결과를 기존 플랫폼에 쉽게 적용할 수 있으며, 이를 통해 빅데이터 분석 플랫폼의 성능 향상에 많은 영향을 끼칠 수 있을 것으로 기대한다.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", ACM Communications of the ACM, Vol. 51, No. 1, pp. 107-113, Jan. 2008

[2] M.. Zaharia, R. S. Xin, and I. Stoica, "Apache Spark: a unified engine for big data processing", Communications of the ACM, Vol. 59, No. 11, pp. 56-65, Nov. 2016.

[3] STORM, A. Storm, "distributed and fault-tolerant realtime computation", https://storm.apache.org/ [accessed: Jan. 12, 2019]

[4] BIGSTREAM, https://bigstream.co/ [accessed: Jan. 12, 2019]

[5] B. Bloom, "Space/time tradeoffs in hash coding with allowable errors", Communications of the ACM, Vol. 13, No. 7, pp. 422-426, Jul. 1970.

[6] James K. Mullin, "Optimal semijoins for distributed database systems", IEEE Transactions on Software Engineering, Vol. 16, No. 5, pp. 558-560, May 1990.

[7] James K. Mullin, "Estimating the size of a relational join", Information Systems, Vol. 18, No. 3, pp. 189-196, Apr. 1993.

[8] S. C. Rhea and J. Kubiatowicz, "Probabilistic Location and Routing", in Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), IEEE Computer Society, Vol. 3, pp. 1248-1257, Jun. 2002.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network. ACM SIGCOMM Computer Communication Review", Proceedings of the 2001 SIGCOMM Conference, Vol. 31, No. 4, pp. 161-172, Oct. 2001.

[10] S. Dharmapurikar, P. Krishnamurthy, T .S. Sproull, and J. W. Lockwood, "Deep packet inspection using Parallel Bloom Filters", IEEE Micro, Vol.

24, No. 1, pp. 52-61, Jan-Feb. 2004.
[11] S. Dharmapurikar and J. W. Lockwood, "Fast and Scalable Pattern Matching for Network Intrusion Detection Systems", IEEE Journal on Selected Areas in Communications, Vol. 24, No. 10, pp. 1781-1792, Oct. 2006.
[12] D. Guo, J. Wu, H. Chen, X., and X. Luo, "The Dynamic Bloom filters", IEEE Transactions on knowledge and data engineering, Vol. 22, No. 1, pp. 120-133, Jan. 2010.
[13] Redis, "In-memory key-value database", https://redis.io. [accessed: Jan. 12, 2019]
[14] Apache Parquet, https://parquet.apache.org/ [accessed: Jan. 04, 2019]
[15] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol", IEEE/ACM Transactions on Networking, Vol, 8, No. 3, pp. 281-293, Jun. 2000.

저자소개

박 홍 규 (Hongkyu Park)

2004년 : 연세대학교 정보산업공학과 학사
2007년 : 연세대학교 대학원 컴퓨터공학과 석사
2012년 : 연세대학교 대학원 컴퓨터과학과 박사
2012년 ~ 2015년 : 삼성전자 책임연구원
2015년 ~ 2017년 : SK텔레콤 종합기술원
2017년 ~ 현재 : 동양미래대학교 조교수
관심 분야 : 실시간 빅데이터 처리 및 분석, NoSQL

박 남 훈 (Nam Hun Park)

2000년 : 연세대학교 컴퓨터과학과 학사
2002년 : 연세대학교 대학원 컴퓨터과학과 석사
2007년 : 연세대학교 대학원 컴퓨터과학 박사
2007년 9월 ~ 2008년 8월 : 연세대학교 데이터베이스 국가지정 연구실 연구원
2008년 9월 ~ 2010년 2월 : Worcester Polytecinic Institute Research Associate
2010년 ~ 현재 : 안양대학교 교수
관심 분야 : 데이터베이스, 빅데이터, 스트림 마이닝