K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Batch:BDA_3 **Roll No.: 121161**

Experiment / assignment / tutorial No: 6

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE : Implementation of any one Data Streaming algorithm using Map Reduce.

AIM : To Implementation of any one Data Streaming algorithm using Map Reduce.

Expected Outcome of Experiment :

CO 3 : Interpret business modes and scientific computing paradigm and apply software tools for Big data analytics.

Books/ Journals/ Websites referred :

https://pig.apache.org/docs/r0.7.0/setup.html

https://archanaschangale.wordpress.com/2013/10/14/pig-installation- on-ubuntu/

http://hadooptutorial.info/pig-installation- on-ubuntu/

Pre Lab/ Prior Concepts :

In a DBMS, input is under the control of the programming staff. ƒ SQL INSERT commands or bulk loaders. ƒ Stream Management is important when the input rate is controlled externally. Example: Google search queries.
Input tuples enter at a rapid rate, at one or more input ports. ƒ The system cannot store the entire stream accessibly.
Types of queries:

1. Ad-hoc queries: Normal queries asked one time about streams. ƒ
   Example: What is the maximum value seen so far in stream S?
2. Standing queries: Queries that are, in principle, asked about the stream at all times.
   Example: Report each new maximum value ever seen in stream S.

Applications:

- Mining query streams. Google wants to know what queries are more frequent today than yesterday. ƒ

- Mining click streams. Yahoo! wants to know which of its pages are getting an unusual number of hits in the past hour. ƒ
- IP packets can be monitored at a switch. Gather information for optimal routing. ƒ Detect denial of service attacks.

Bloom filter:

To motivate the Bloom filter idea, consider a web crawler. ƒ It keeps, centrally, a list of all the URL's it has found so far. ƒ It assigns these URL's to any of a number of parallel tasks; these tasks stream back the URL's they find in the links they discover on a page. ƒ It needs to filter out those URL's it has seen before.

A Bloom filter is an array of bits, together with a number of hash functions. ƒ The argument of each hash function is a stream element, and it returns a position in the array. ƒ Initially, all bits are 0. ƒ When input x arrives, we set to 1 the bits h(x), for each hash function h.

BloomsFilter Code:

```
import java.util.*;
import java.io.*;
class bloom
{
static int size=0;
public static void main(String args[])
{
      Scanner sc=new Scanner(System.in);
      System.out.println("Size of Array");
      int e=0;
      size=sc.nextInt();
       byte arr[]=new byte[size];
      for(int i=0;i<size;i++)
            arr[i]=0;

      do
      {
      System.out.println("Enter your choice:");
      System.out.println("1) Insert");
      System.out.println("2) Check");
      int c=sc.nextInt();
      switch(c)
      {
      case 1:
      System.out.println("Enter the number to be inserted");
      int n=sc.nextInt();
      String x=insert(n);
      String s1[]=x.split(" ");
      arr[Integer.parseInt(s1[0])]=1;
      arr[Integer.parseInt(s1[1])]=1;
      for(byte i:arr)
            System.out.print(i+" ");
break;
      case 2:
```

```
      check(arr);
      break;
      }
System.out.println("\nEnter 3 to Continue\nEnter 4 to Exit");
 e=sc.nextInt();
      }while(e!=4);
}
public static String insert(int n)
{
      String o="";
      String e="";
      String s=Integer.toBinaryString(n);
      char a[]=s.toCharArray();
      for(int i=0;i<a.length;i=i+2)
      {
            e+=a[i];
      }
      for(int i=1;i<a.length;i=i+2)
      {
            o+=a[i];
      }
            int ei=Integer.parseInt(e,2)%size;
            int oi=Integer.parseInt(o,2)%size;

            return ei+" "+oi;
}
public static void check(byte arr[]){
System.out.println("\nEnter the number to be checked");
Scanner sc= new Scanner(System.in);
int n=sc.nextInt();
String ch=insert(n);
String s1[]=ch.split(" ");
if(arr[Integer.parseInt(s1[0])]==1&&arr[Integer.parseInt(s1[1])]==1)
      System.out.println("Present");
else
      System.out.println("absent");
}

}
```

Output:

```
mit@mit-Inspiron-3542:~/Desktop$ sudo javac bloom.java

mit@mit-Inspiron-3542:~/Desktop$ java bloom

Size of Array

11

Enter your choice:

1) Insert

2) Check
```

```
1

Enter the number to be inserted

118

0 0 0 1 0 1 0 0 0 0 0

Enter 3 to Continue

Enter 4 to Exit

3

Enter your choice:

1) Insert

2) Check

1

Enter the number to be inserted

4

1 0 1 1 0 1 0 0 0 0 0

Enter 3 to Continue

Enter 4 to Exit

3

Enter your choice:

1) Insert

2) Check

2

Enter the number to be checked

118

Present

Enter 3 to Continue

Enter 4 to Exit

4
```

Conclusion :

Thus we studied and implemented bloom filter data streaming algorithm using map reduce.

Post Lab Questions:

1. Explain the DGIM algorithm with an example.

The algorithm uses O(log2 N) bits to represent a window of N bits, and allows us to estimate the number of 1's in the window with an error of no more than 50%. To begin, each bit of the stream has a timestamp, the position in which it arrives. The first bit has timestamp 1, the second has timestamp 2, and so on.
Since we only need to distinguish positions within the window of length N, we shall represent timestamps modulo N, so they can be represented by log2 N bits. If we also store the total number of bits ever seen in the stream (i.e., the most recent timestamp) modulo N, then we can determine from a timestamp modulo N where in the current window the bit with that timestamp is. We divide the window into buckets,5 consisting of:

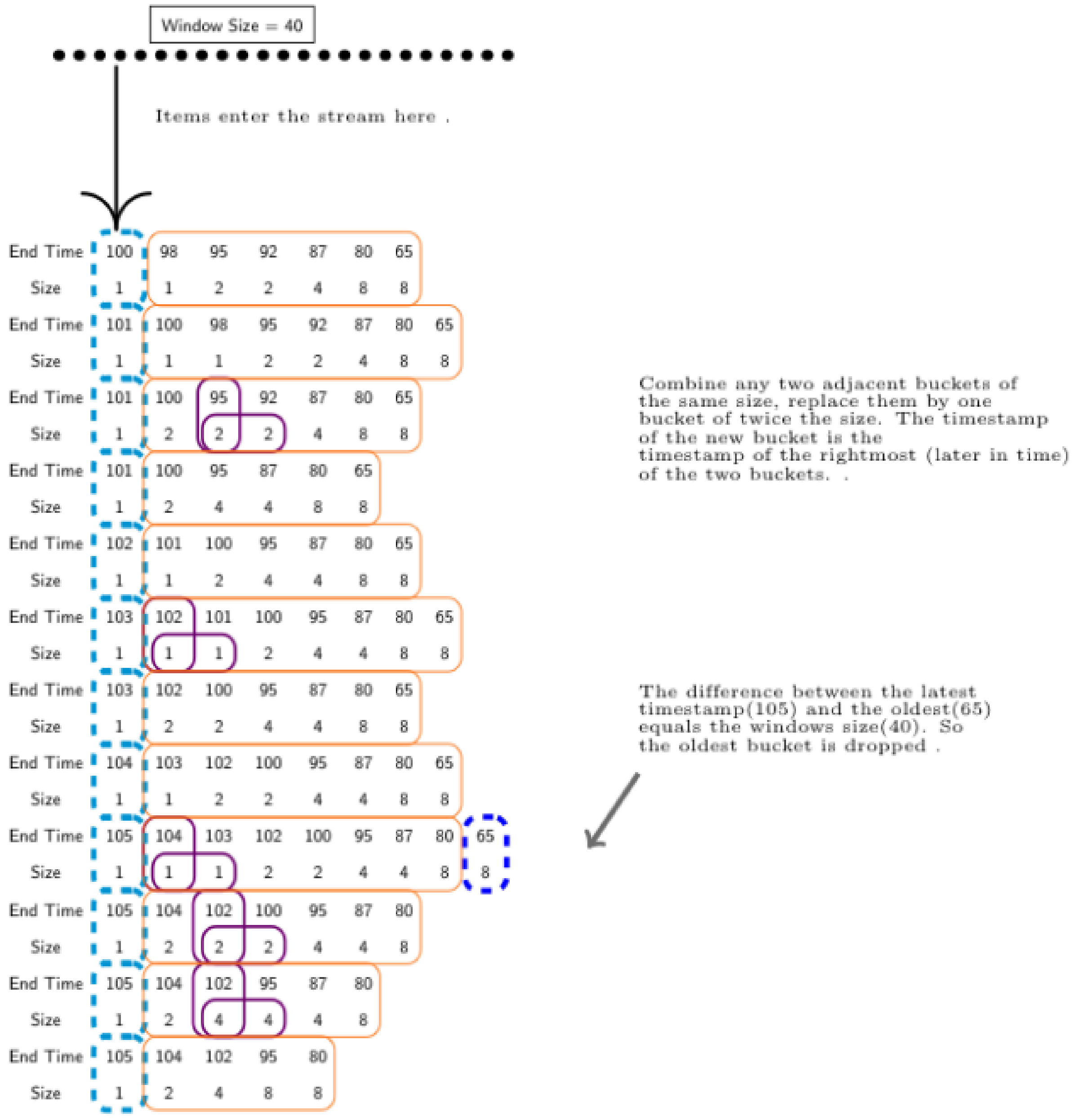
1. The timestamp of its right (most recent) end.
2. The number of 1's in the bucket. This number must be a power of 2, and we refer to the number of 1's as the size of the bucket.

To represent a bucket, we need log2 N bits to represent the timestamp (modulo N) of its right end. To represent the number of 1's we only need log2 log2 N bits. The reason is that we know this number i is a power of 2, say 2j , so we can represent i by coding j in binary. Since j is at most log2 N, it requires log2 log2 N bits. Thus, O(logN) bits suffice to represent a bucket. There are six rules that must be followed when representing a stream by buckets.

- The right end of a bucket is always a position with a 1.
- Every position with a 1 is in some bucket.
- No position is in more than one bucket.
- There are one or two buckets of any given size, up to some maximum size.
- All sizes must be a power of 2.
- Buckets cannot decrease in size as we move to the left (back in time).

Example:
Suppose we are using the DGIM algorithm of Section 4.6.2 to estimate the number of 1's in suffixes of a sliding window of length 40. The current timestamp is 100.

Note: we are showing timestamps as absolute values, rather than modulo the window size, as DGIM would do.
Suppose that at times 101 through 105, 1's appear in the stream. Compute the set of buckets that would exist in the system at time 105.


Window Size = 40
Items enter the stream here .
End Time 100 98 95 92 87 80 65
Size 1 1 2 2 4 8 8
End Time 101 100 98 95 92 87 80 65
Size 1 1 1 2 2 4 8 8
End Time 101 100 95 92 87 80 65
Size 1 2 2 2 4 8 8
End Time 101 100 95 87 80 65
Size 1 2 4 4 8 8
End Time 102 101 100 95 87 80 65
Size 1 1 2 4 4 8 8
End Time 103 102 101 100 95 87 80 65
Size 1 1 1 2 4 4 8 8
End Time 103 102 100 95 87 80 65
Size 1 2 2 4 4 8 8
End Time 104 103 102 100 95 87 80 65
Size 1 1 2 2 4 4 8 8
End Time 105 104 103 102 100 95 87 80 65
Size 1 1 1 2 2 4 4 8 8
End Time 105 104 102 100 95 87 80
Size 1 2 2 2 4 4 8
End Time 105 104 102 95 87 80
Size 1 2 4 4 4 8
End Time 105 104 102 95 80
Size 1 2 4 8 8
Combine any two adjacent buckets of the same size, replace them by one bucket of twice the size. The timestamp of the new bucket is the timestamp of the rightmost (later in time) of the two buckets. .
The difference between the latest timestamp(105) and the oldest(65) equals the windows size(40). So the oldest bucket is dropped .