Tutorial 9: Stream Processing

Li Yuan

li.yuan@u.nus.edu

Reservoir Sampling

- **Task: select *s* elements from a stream of size *N* with uniform probability**

- **Solution: Reservoir sampling**
  - Store first *s* elements
  - For the *k*-th element thereafter, keep with probability *s*/*k* (randomly discard an existing element)

Reservoir Sampling

- **Example: s = 10**
  - Keep first 10 elements
  - 11th element: keep with 10/11

    If we decide to keep it: sampled uniformly by definition

    →probability existing item discarded: 10/11 × 1/10 = 1/11

    →probability existing item survives: 10/11. done.

Problem 1

The reservoir sampling algorithm maintains a sample S (of s elements) with the desired property: After n elements, the sample contains each element seen so far with probability s/n. Prove this property. [Hints: We prove this by induction.]

Solution 1

We prove this by induction:

- Assume that after ***n*** elements, the sample contains each element seen so far with probability ***s/n***
- We need to show that after seeing element ***n+1*** the sample maintains the property
  - **Sample S contains each element seen so far with probability *s/(n+1)***

Base case:

- After we see **n=s** elements the sample **S** has the desired property
  - **Each out of n=s elements is in the sample with probability *s/s = 1***

Solution 1

- **Inductive hypothesis: After *n* elements, the sample *S* contains each element seen so far with prob. *s/n***
- **Now element *n+1* arrives**
- **Inductive step:**
  - So, at time ***n***, tuples in ***S*** were there with prob. **s/n**
  - Time ***n→n+1***, tuple stayed in ***S*** with prob. **n/(n+1)** $\left(1 - \frac{s}{n+1} * \frac{1}{s} = \frac{n}{n+1}\right)$
  - So prob. tuple is in ***S*** at time ***n+1*** = $\frac{s}{n} \cdot \frac{n}{n+1} = \frac{s}{n+1}$

Bloom Filters

Task: keep track of set membership

put(x) → insert x into the set

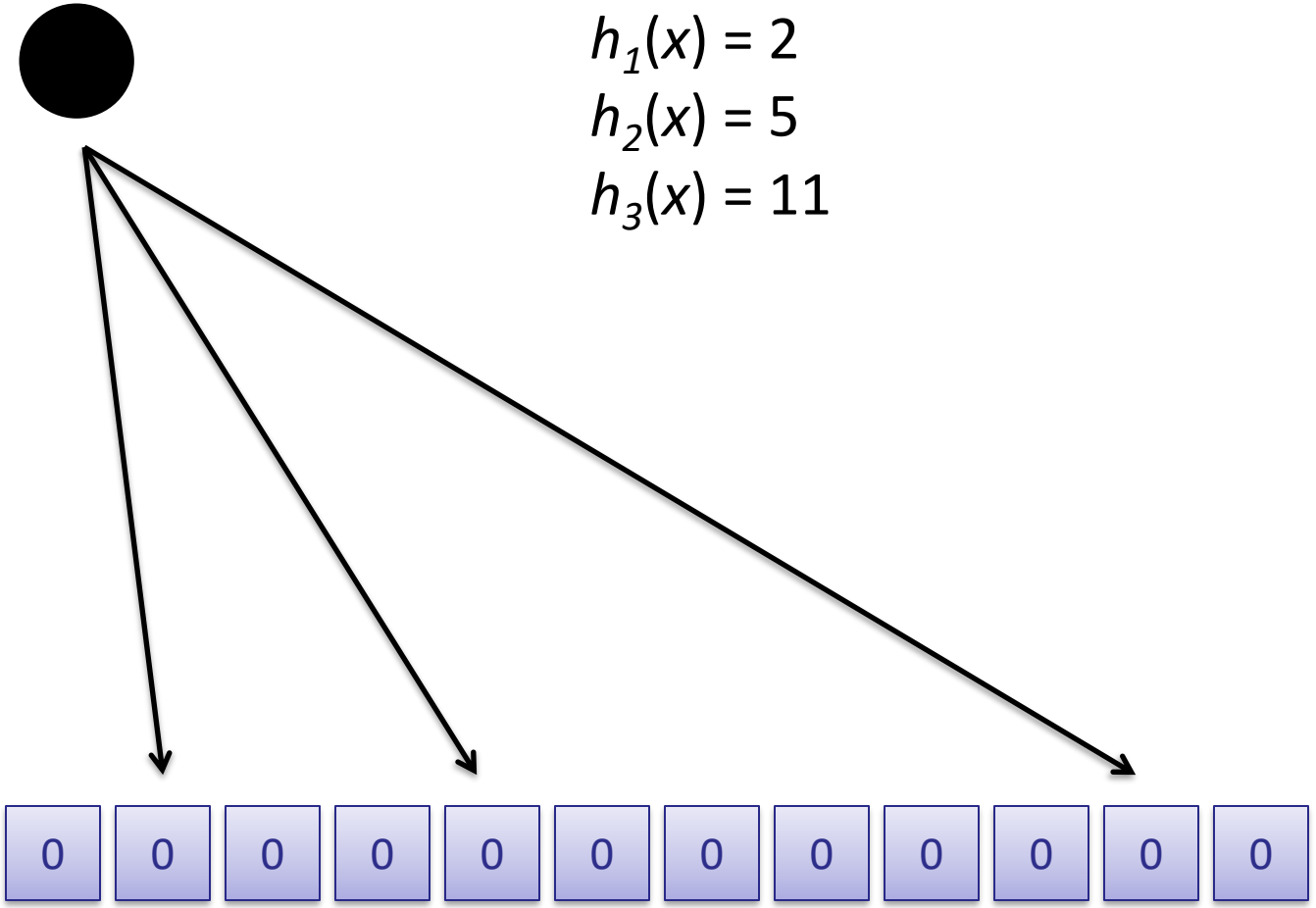
contains(x) → yes if x is a member of the set

Components

m-bit bit vector

k hash functions: $h_1 \ldots h_k$

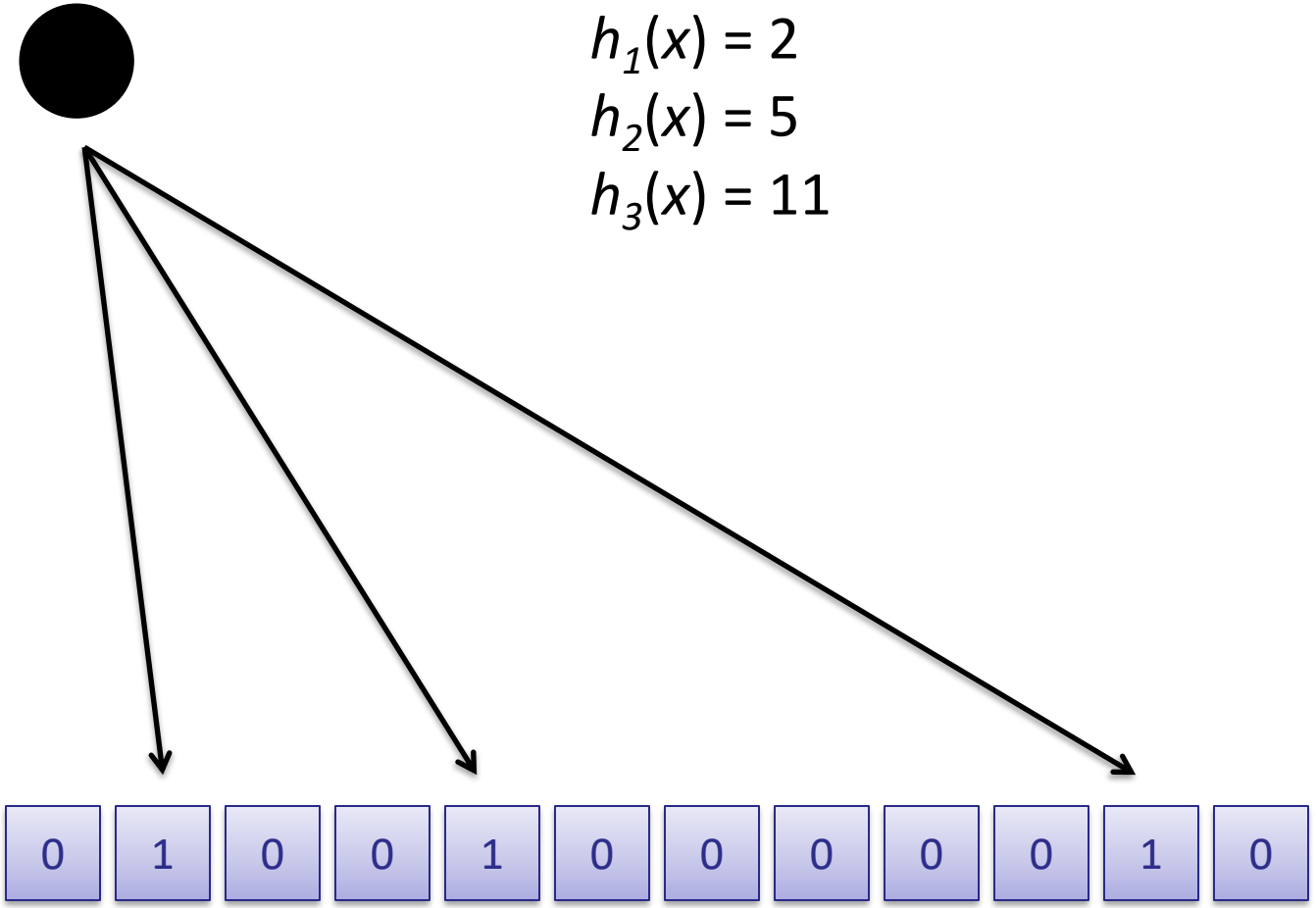
Bloom Filters: put



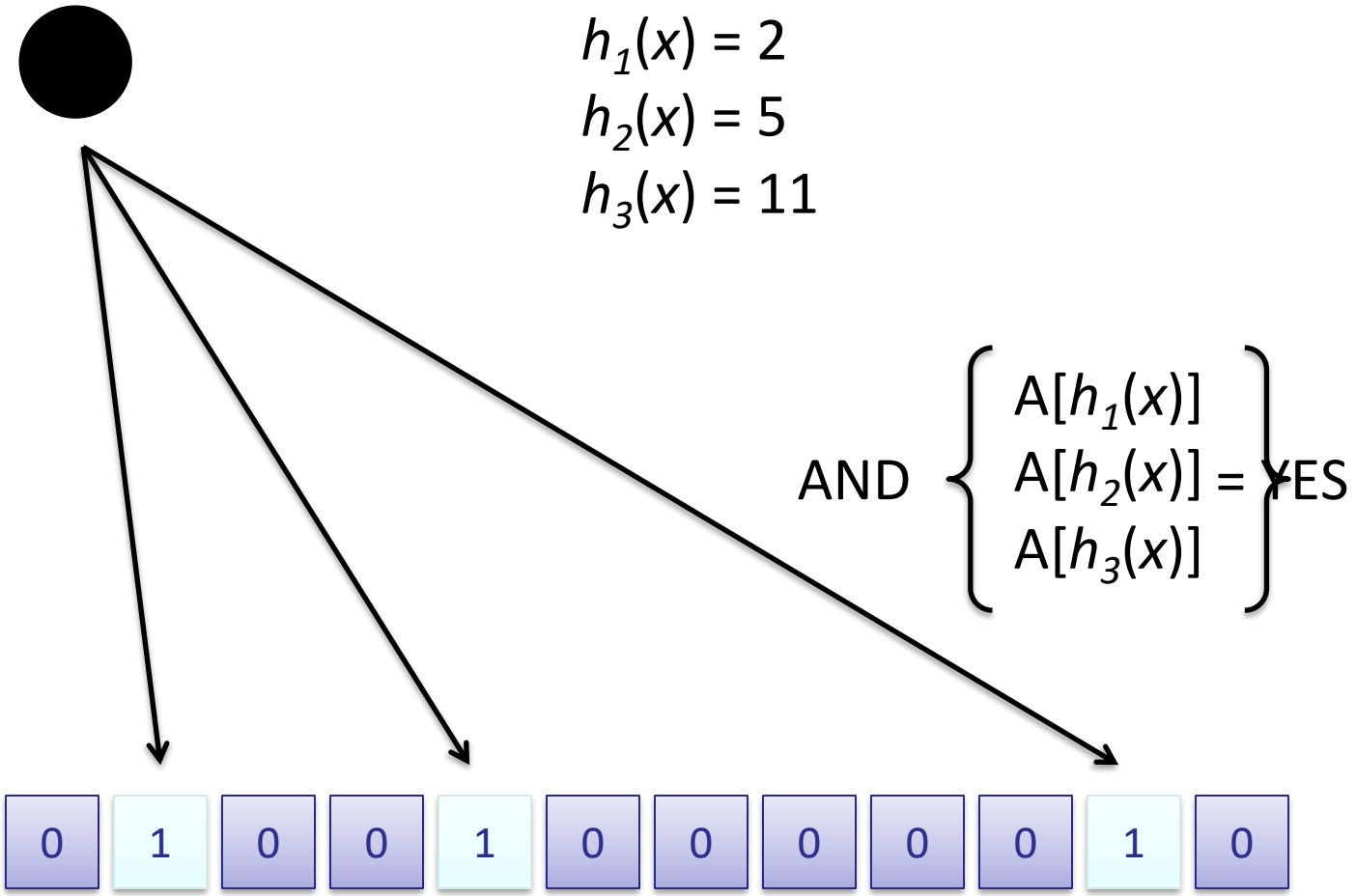
Bloom Filters: put

0	1	0	0	1	0	0	0	0	0	1	0

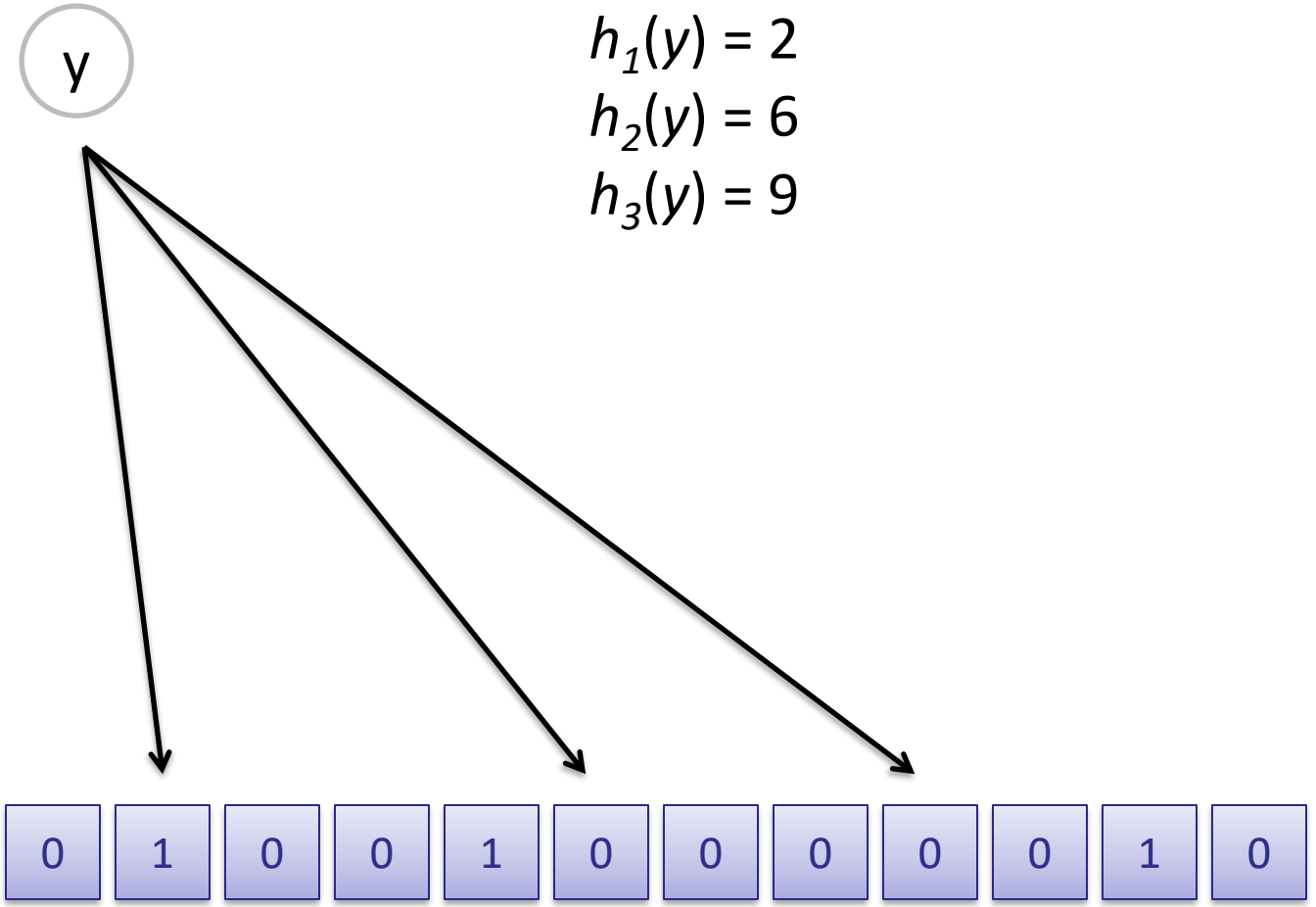
Bloom Filters: contains



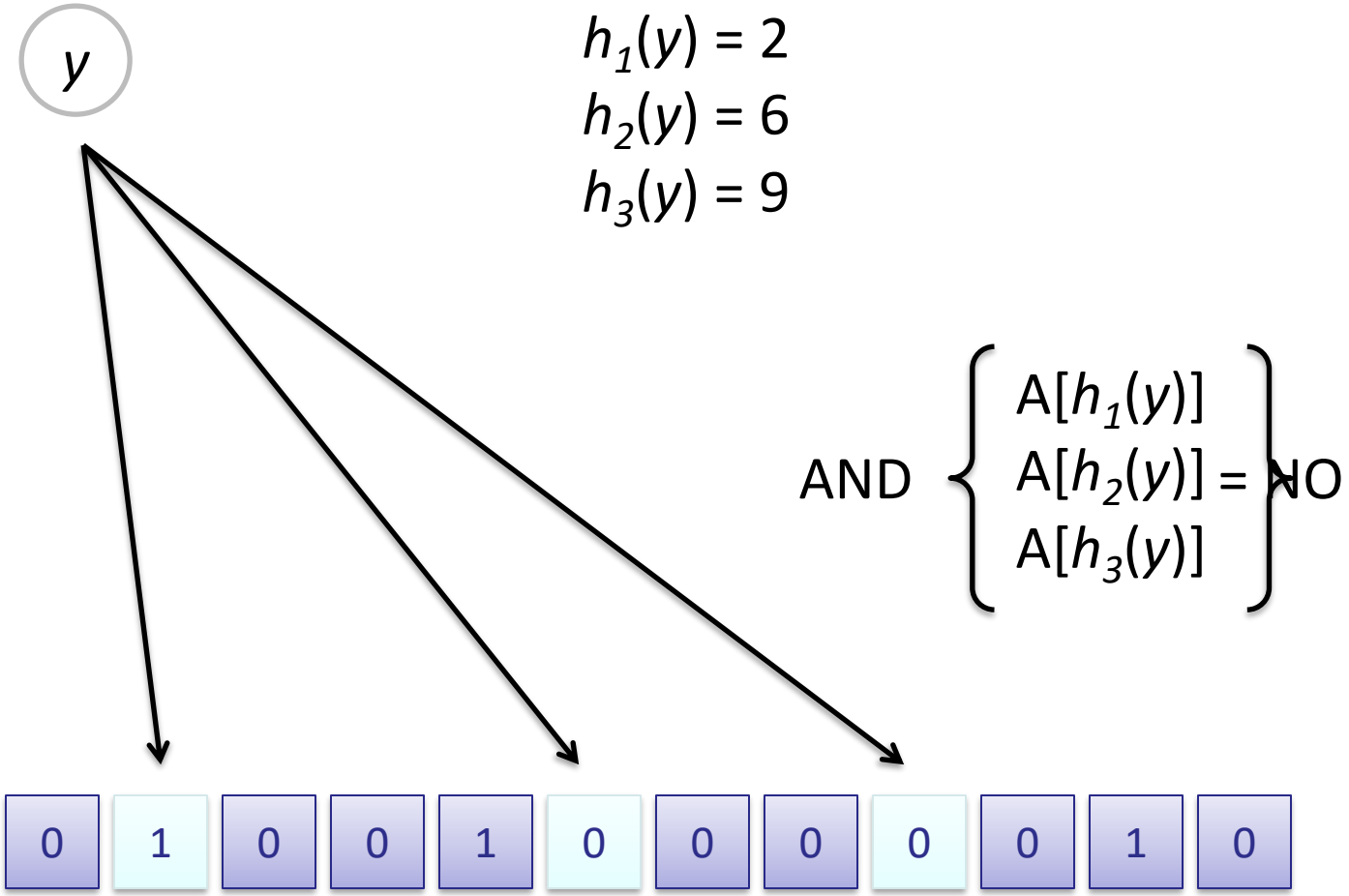
Bloom Filters: contains



Bloom Filters: contains



Bloom Filters: contains



Problem 2

Consider a bloom filter with an m-bit vector, and k hash functions: h_1, h_2, …, h_k. What is the false positive probability for a membership operation?

False positive error:
A false positive error, or in short a false positive, commonly called a "false alarm", is a result that indicates a given condition exists, when it does not.

False negative error:
A false negative error, or in short a false negative, is a test result that indicates that a condition does not hold, while in fact it does.

Solution 2

- **The probability that a specific bit is still 0 after a hash:**
  - $p_1 = 1 - \frac{1}{m}$
- **The probability that a specific bit is still 0 after a member has been hashed:**
  - $p_2 = \left(1 - \frac{1}{m}\right)^k$
- **The probability that a specific bit is still 0 after all members of S (the size is n) have been hashed:**
  - $p = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$
- **For a non member, it may be found to be a member of S (all of its k bits are nonzero) with false positive probability**
  - $(1 - p)^k$

Flajolet-Martin (FM) Counter

- **Pick a hash function *h* that maps each of the *N* elements to at least $\log_2 N$ bits**

- **For each stream element *a*, let *r*(*a*) be the number of trailing 0s in *h*(*a*)**
  - **r(a)** = position of first 1 counting from the right
    - **E.g., say *h(a)* = *12*, then *12* is *1100* in binary, so *r(a)* = *2***

- **Record *R* = the maximum *r*(*a*) seen**
  - **$R = \max_a r(a)$**, over all the items ***a*** seen so far

- **Estimated number of distinct elements = 2^R**

Problem 3

We wish to use Flajolet-Martin counter algorithm to count the number of distinct elements in a stream. Suppose that there are ten possible elements, 1, 2, …, 10, that could appear in the stream, but only four of them have actually appeared. To make our estimation of the count of distinct elements, we hash each element to a 4-bit binary number. The element x is hashed to 3x + 7 (modulo 11). For example, element 8 hashes to 3*8+7 = 31, which is 9 modulo 11 (i.e., the remainder of 31/11 is 9). Thus, the 4-bit string for element 8 is 1001.

Problem 3

A set of four of the elements 1 through 10 could give an estimate that is exact (if the estimate is 4), or too high, or too low. You should figure out under what circumstances a set of four elements falls into each of those categories. Identify the set of four elements that gives the exactly correct estimate.

Solution 3

x	3x+7	Mod 11 (in binary form)
1	10	1010
2	13	0010
3	16	0101
4	19	1000
5	22	0000
6	25	0011
7	28	0110
8	31	1001
9	34	0001
10	37	0100

Solution 3

In order to give the correct estimate (4), a set must have at most two 0's at the end of the hash value of any of its members, but must have a member with exactly two 0's at the end. Observe from the table above that 10 is the only element whose hash value has exactly two bits at the end. However, 1, 2, 3, 6, 7, 8, and 9 have zero or one 0 at the end, so the correct answers are any set of four elements that includes 10 and does not include 4 or 5.

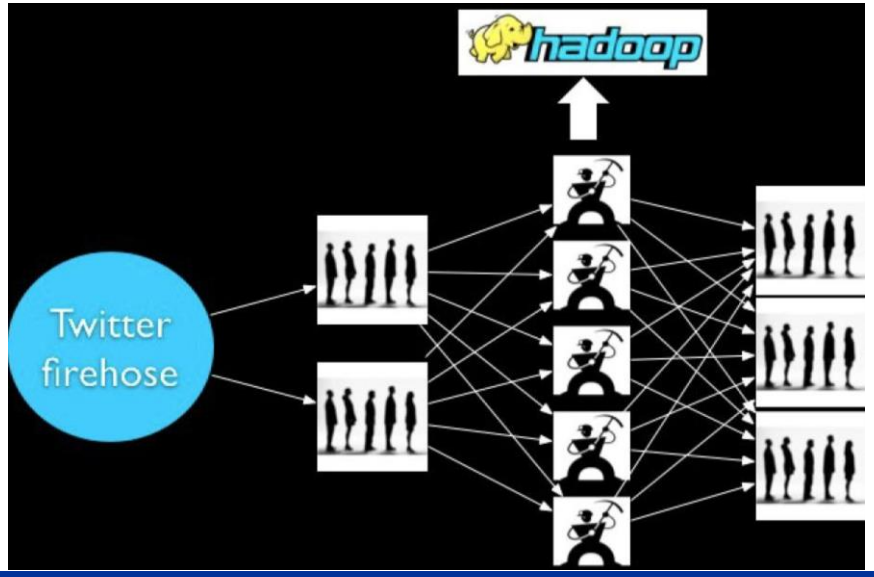
Problem 4

Consider Storm architecture and the job topologies in tweet processing in Twitter. Prior to Storm, the task of "schemify tweets and append to Hadoop" is implemented in Hadoop, as illustrated in the below picture. There are three major issues of implementing the task using Hadoop. Explain the reasons.

a) Scaling is painful

b) Poor fault-tolerance

c) Coding is tedious



Solution 4

a) Scaling is painful: the new data comes in continuously, and it does not meet the batch nature of Hadoop.

b) Poor fault-tolerance: streaming can be stateful.

c) Coding is tedious: Map/Reduce is too restrict in expressing various streaming logics.

Acknowledgement

Thanks to Li Qinbin for making these slides.

liqinbin@u.nus.edu